Object Oriented Programming Using

C#

By Naveed Zaman

Object Oriented Programming Using C#

This free book is provided by courtesy of [C# Corner] and Mindcracker Network and its authors. Feel free to share this book with your friends and co-workers.

Please do not reproduce, republish, edit or copy this book.

Naveed Zaman
Senior Software Engineer and Database Developer

Sam Hobbs
Editor, C# Corner

This book is a basic introduction to "**Object Oriented Programming Using C#**" for beginners who have never used C# before. After completing this book, you will understand:

- Basic introduction to C# Classes
- C# Class Properties
- Use of Constructor
- OOP's Concepts like encapsulation, polymorphism etc.
- Interface & Delegates

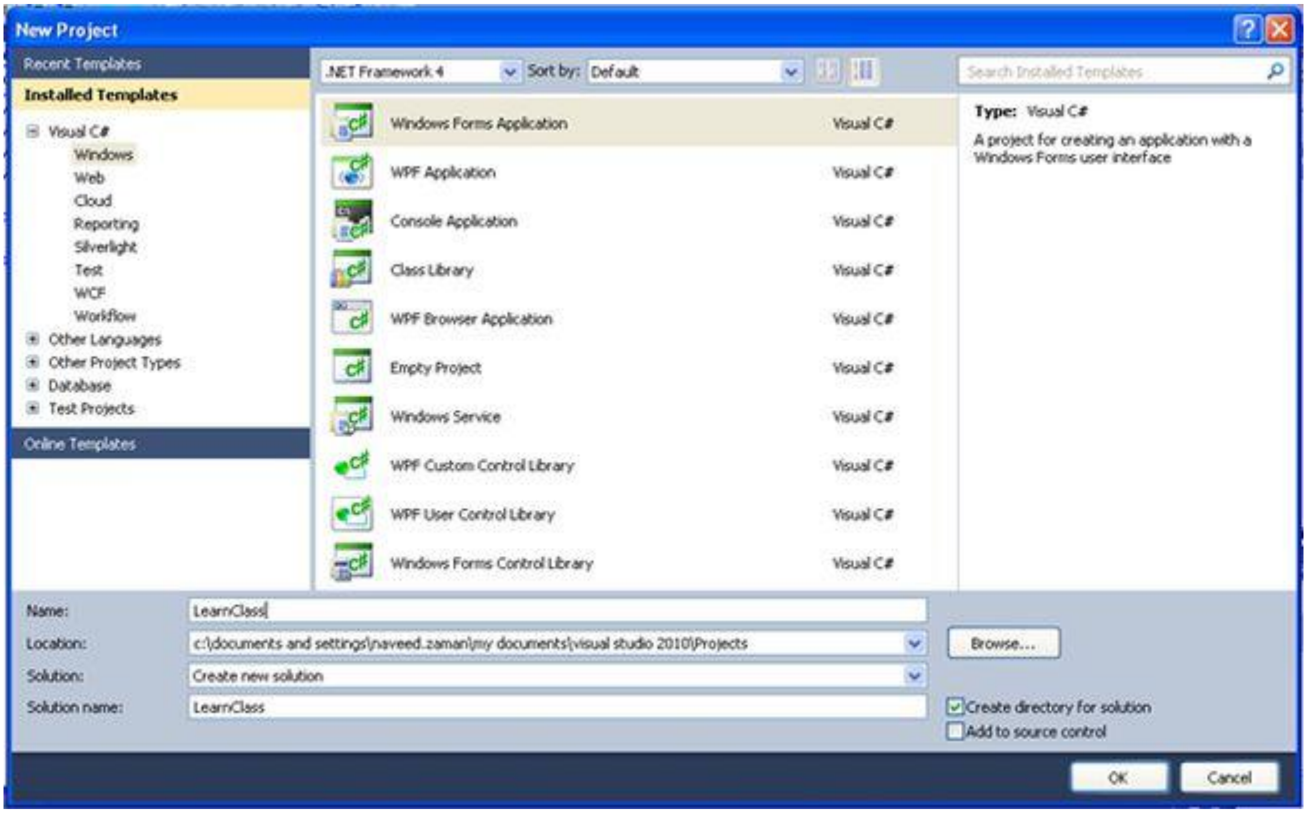
Table of Contents

1. Introduction to C# Classes

C# is totally based on Object Oriented Programming (OOP). First of all, a class is a group of similar methods and variables. A class contains definitions of variables, methods etcetera in most cases. When you create an instance of this class it is referred to as an object. On this object, you use the defined methods and variables.

Step 1

Create the new project with name "LearnClass" as shown below.


New Project
Recent Templates
Installed Templates
Visual C#
Windows
Web
Cloud
Reporting
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Database
Test Projects
Online Templates
Sort by:
Search Installed Templates
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Windows Service Visual C#
WPF Custom Control Library Visual C#
WPF User Control Library Visual C#
Windows Forms Control Library Visual C#
Browse...
OK
Cancel

.NET Framework 4

Sort by: Default

Type: Visual C#

A project for creating an application with a Windows Forms user interface

Name: LearnClass

Location: c:\documents and settings\naveed.zaman\my documents\visual studio 2010\Projects

Solution: Create new solution

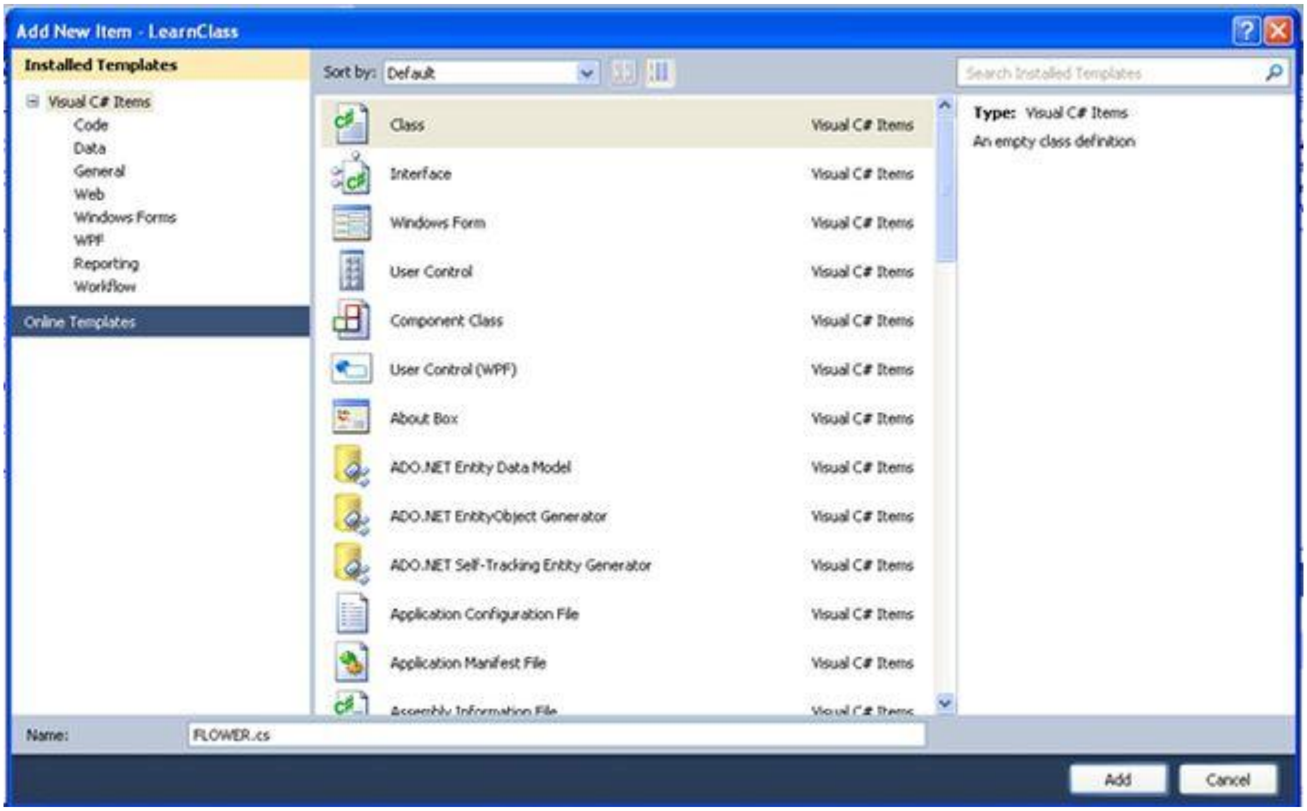
Solution name: LearnClass

☑ Create directory for solution

□ Add to source control

Step 2

Using "Project" -> "Add Class" we add a new class to our project with the name "FLOWER".



Sort by: Default

Class	Visual C# Items
Interface	Visual C# Items
Windows Form	Visual C# Items
User Control	Visual C# Items
Component Class	Visual C# Items
User Control (WPF)	Visual C# Items
About Box	Visual C# Items
ADO.NET Entity Data Model	Visual C# Items
ADO.NET EntityObject Generator	Visual C# Items
ADO.NET Self-Tracking Entity Generator	Visual C# Items
Application Configuration File	Visual C# Items
Application Manifest File	Visual C# Items
Assembly Information File	Visual C# Items

Type: Visual C# Items
An empty class definition

Name: FLOWER.cs

Step 3

Now add the following code in the FLOWER class:

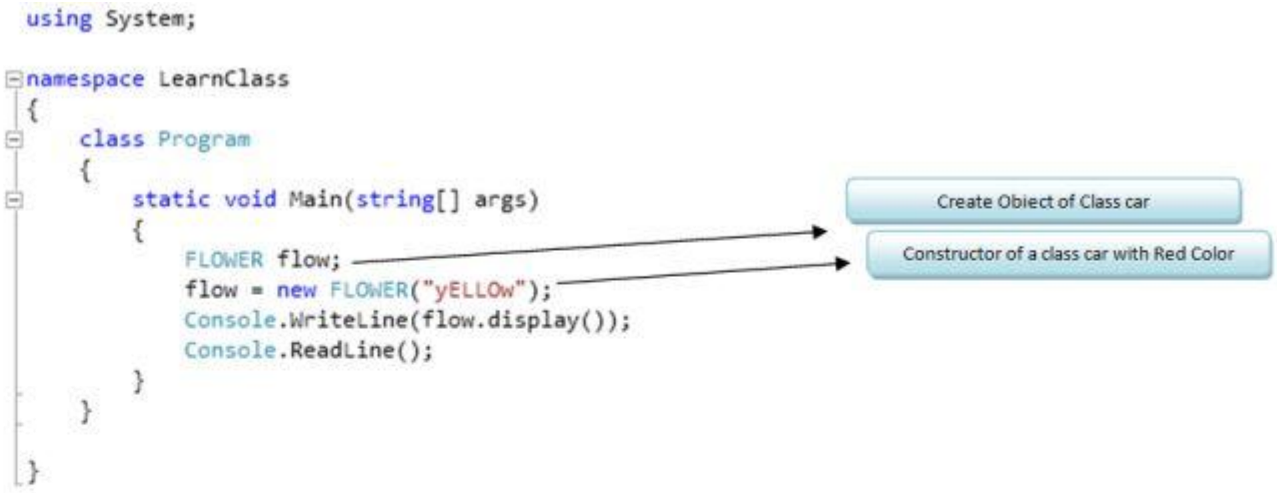
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LearnClass
{
    class FLOWER
    {
        public string flowercolor;
        public FLOWER(string color)
        {
            this.flowercolor = color;
        }
        public string display()
        {
            return "Color of the flower : " + this.flowercolor;
        }

    }
}
```

In this example we have created the class FLOWER. In this class we declared one public variable flowercolor. Our FLOWER class defines a constructor. It takes a parameter that allows us to initialize FLOWER objects with a color. In our case we initialize it with the color yellow. The Describe() method shows the message. It simply returns a string with the information we provided.

Step 4

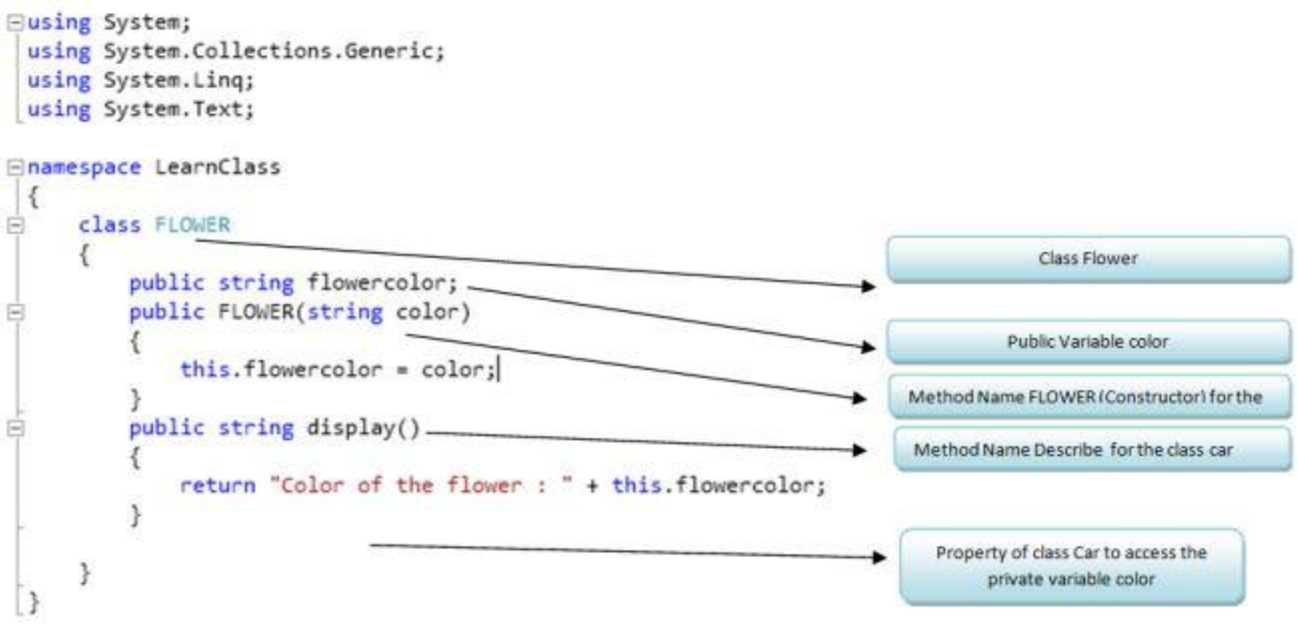
Insert the following code in the Main module.

```
using System;

namespace LearnClass
{
    class Program
    {
        static void Main(string[] args)
        {
            FLOWER flow;
            flow = new FLOWER("YELLOW");
            Console.WriteLine(flow.display());
            Console.ReadLine();
        }
    }
}
```



```
using System;

namespace LearnClass
{
    class Program
    {
        static void Main(string[] args)
        {
            FLOWER flow;
            flow = new FLOWER("yELLOw");
            Console.WriteLine(flow.display());
            Console.ReadLine();
        }
    }

}
```

Create Object of Class car
Constructor of a class car with Red Color



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LearnClass
{
    class FLOWER
    {
        public string flowercolor;
        public FLOWER(string color)
        {
            this.flowercolor = color;
        }
        public string display()
        {
            return "Color of the flower : " + this.flowercolor;
        }

    }
}
```

Class Flower
Public Variable color
Method Name FLOWER (Constructor) for the
Method Name Describe for the class car
Property of class Car to access the private variable color

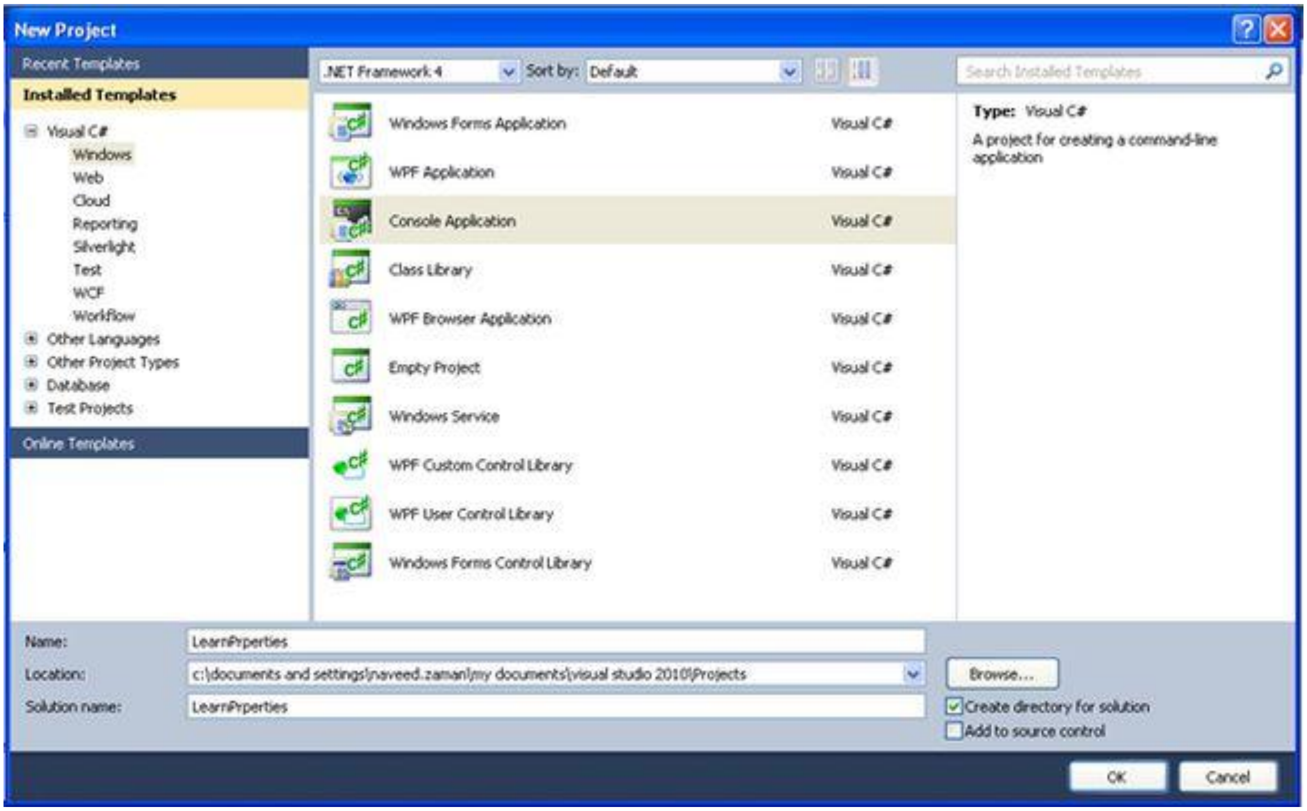
Note: Method name FLOWER and class FLOWER are the same name but the method is a constructor of the class. (We will discuss that topic in detail later on.)

2. Introduction to C# Class Properties

Properties play a vital role in Object Oriented Programming. They allow us to access the private variables of a class from outside the class. It is good to use a private variable in a class. Properties look like a combination of variables and methods. Properties have sections: "a get and a set" method .The get method should return the variable, while the set method should assign a value to it.

Step 1

Open a new project with the name "LearnProperties" as in the following:


New Project
Recent Templates
Installed Templates
Visual C#
Windows
Web
Cloud
Reporting
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Database
Test Projects
Online Templates
.NET Framework 4
Sort by: Default
Search Installed Templates
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Windows Service Visual C#
WPF Custom Control Library Visual C#
WPF User Control Library Visual C#
Windows Forms Control Library Visual C#
Browse...
OK
Cancel

Type: Visual C#

A project for creating a command-line application

Name: LearnPrperties

Location: c:\documents and settings\naveed.zaman\my documents\visual studio 2010\Projects

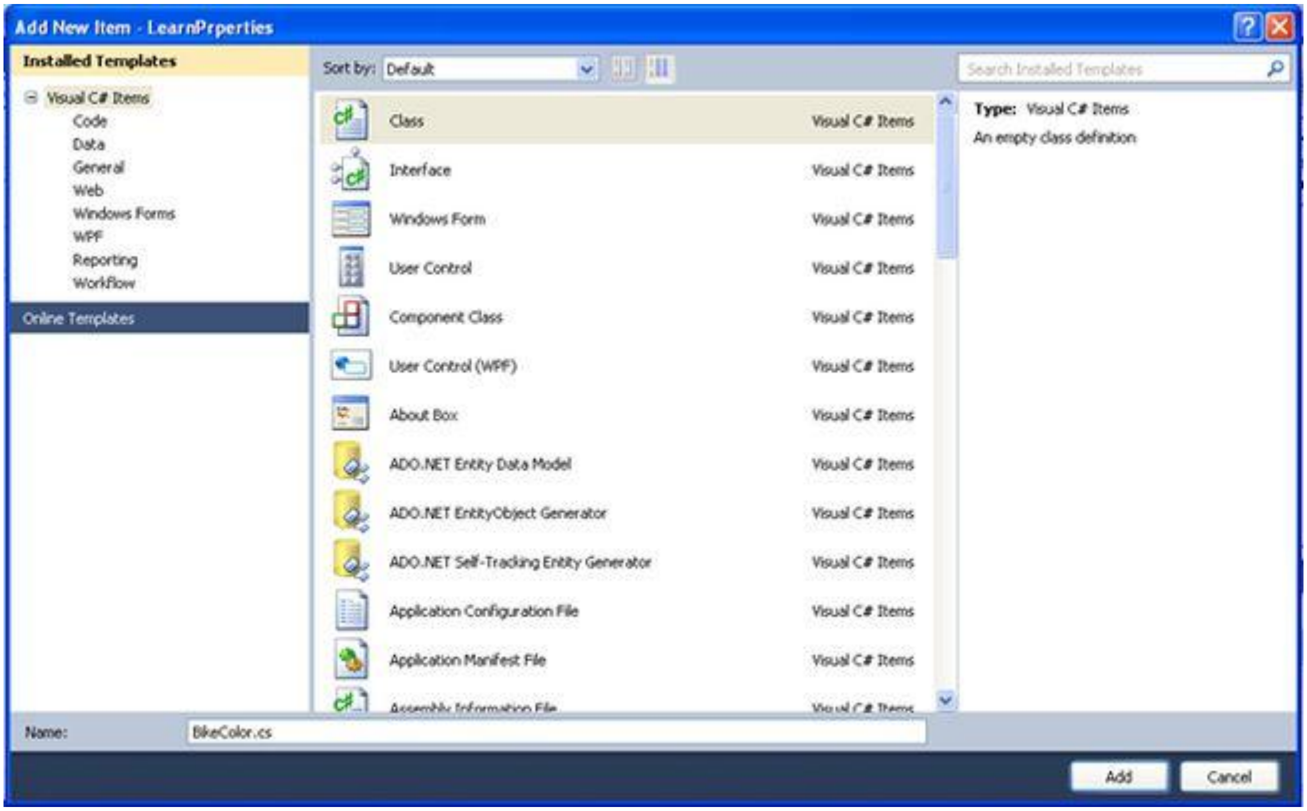
Solution name: LearnPrperties

☑ Create directory for solution

□ Add to source control

Step 2

Now to add the new classes to the project use "Project" -> "Add class" with the class name "BikeColor" as in the following:



Add New Item - LearnPrperties

Sort by: Default

Type: Visual C# Items

An empty class definition

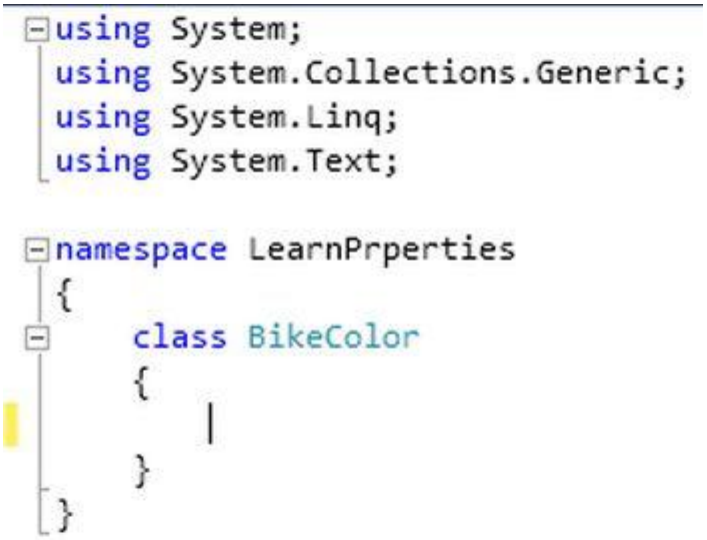
Name: BikeColor.cs

Installed Templates
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
Reporting
Workflow
Online Templates
Search Installed Templates
Class Visual C# Items
Interface Visual C# Items
Windows Form Visual C# Items
User Control Visual C# Items
Component Class Visual C# Items
User Control (WPF) Visual C# Items
About Box Visual C# Items
ADO.NET Entity Data Model Visual C# Items
ADO.NET EntityObject Generator Visual C# Items
ADO.NET Self-Tracking Entity Generator Visual C# Items
Application Configuration File Visual C# Items
Application Manifest File Visual C# Items
Assembly Information File Visual C# Items
Add
Cancel

.

Step 3

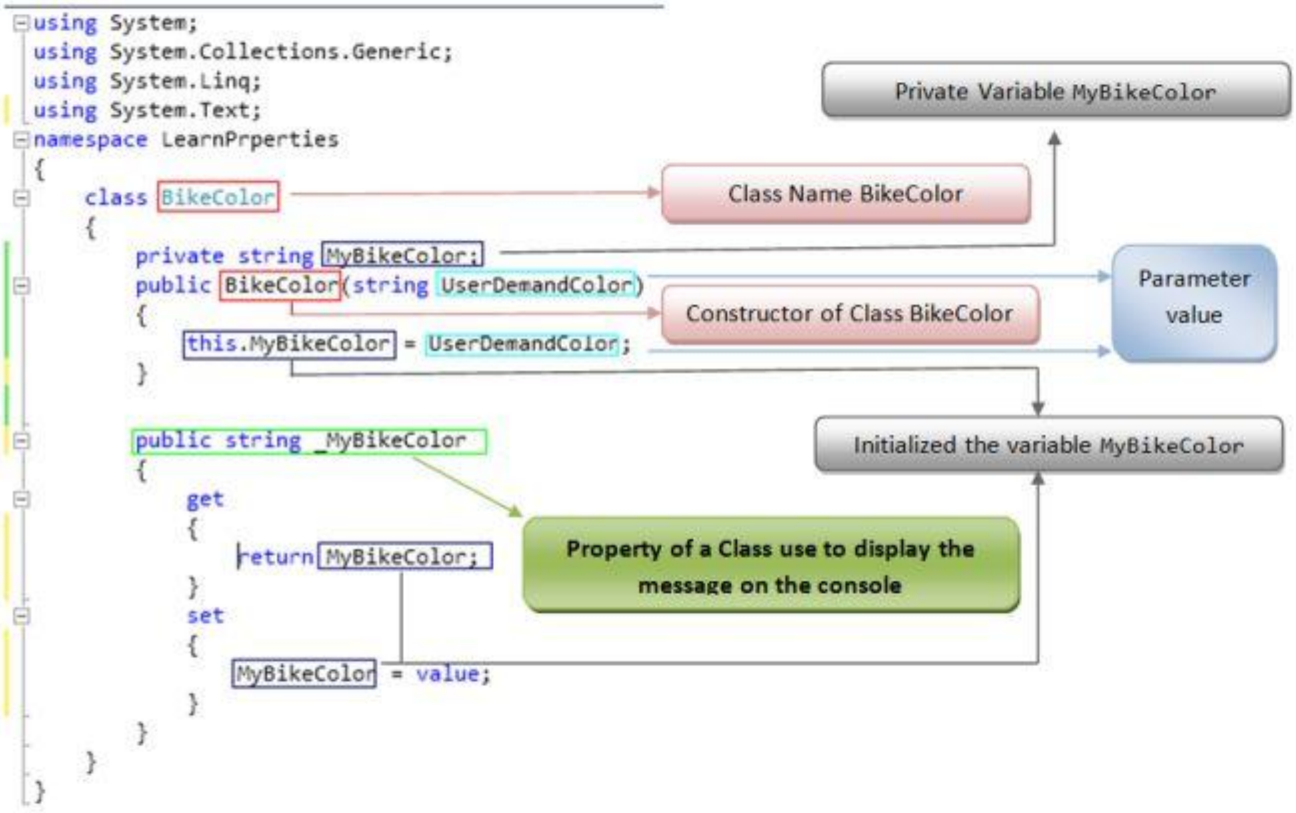
After adding the new class your codes looks like the following:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LearnPrperties
{
    class BikeColor
    {
        |
    }
}
```

Step 4

Insert the following code in the class BikeColor as in the following:

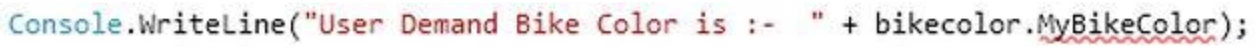


```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace LearnPrperties
{
    class BikeColor
    {
        private string MyBikeColor;
        public BikeColor(string UserDemandColor)
        {
            this.MyBikeColor = UserDemandColor;
        }

        public string _MyBikeColor
        {
            get
            {
                return MyBikeColor;
            }
            set
            {
                MyBikeColor = value;
            }
        }
    }
}
```

Private Variable MyBikeColor
Class Name BikeColor
Parameter value
Constructor of Class BikeColor
Initialized the variable MyBikeColor
Property of a Class use to display the message on the console

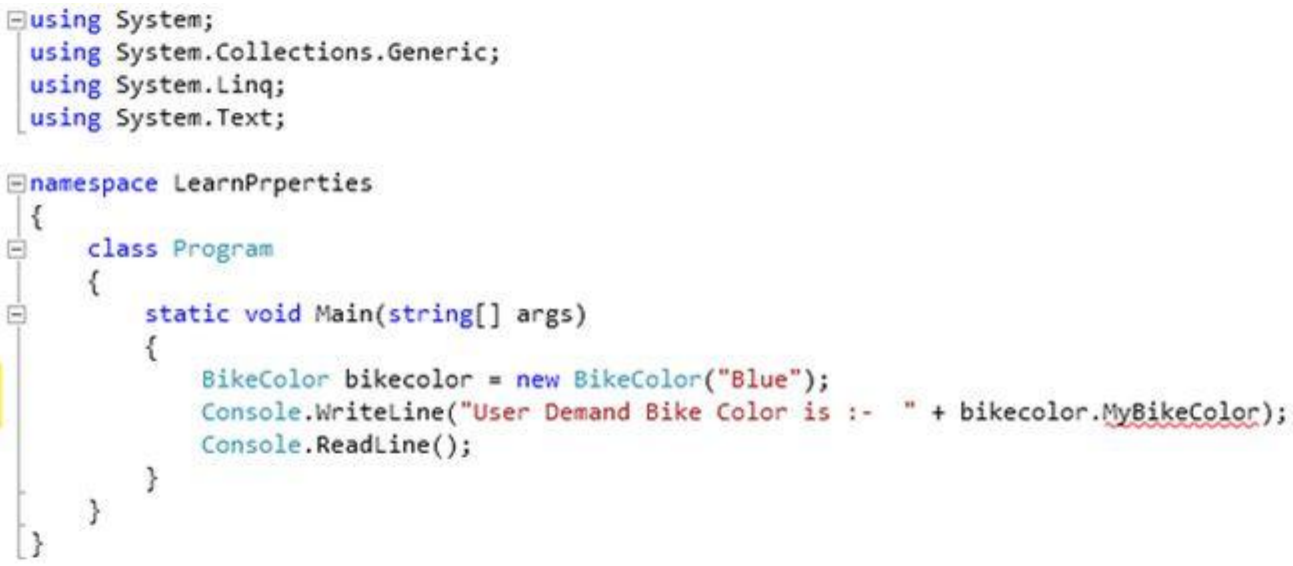
Step 5

Insert the following code in the Main Module. After adding the following code it will show the error.



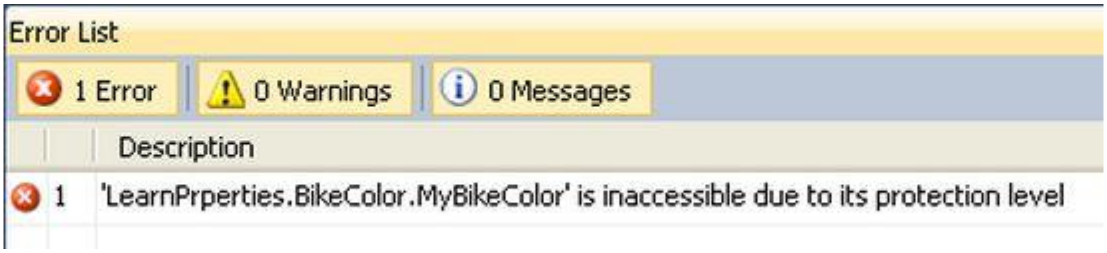
```
Console.WriteLine("User Demand Bike Color is :-  " + bikecolor.MyBikeColor);
```

This is because you are accessing the private variable from outside the class; that is not allowed in OOP.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LearnPrperties
{
    class Program
    {
        static void Main(string[] args)
        {
            BikeColor bikecolor = new BikeColor("Blue");
            Console.WriteLine("User Demand Bike Color is :-  " + bikecolor.MyBikeColor);
            Console.ReadLine();
        }
    }
}
```

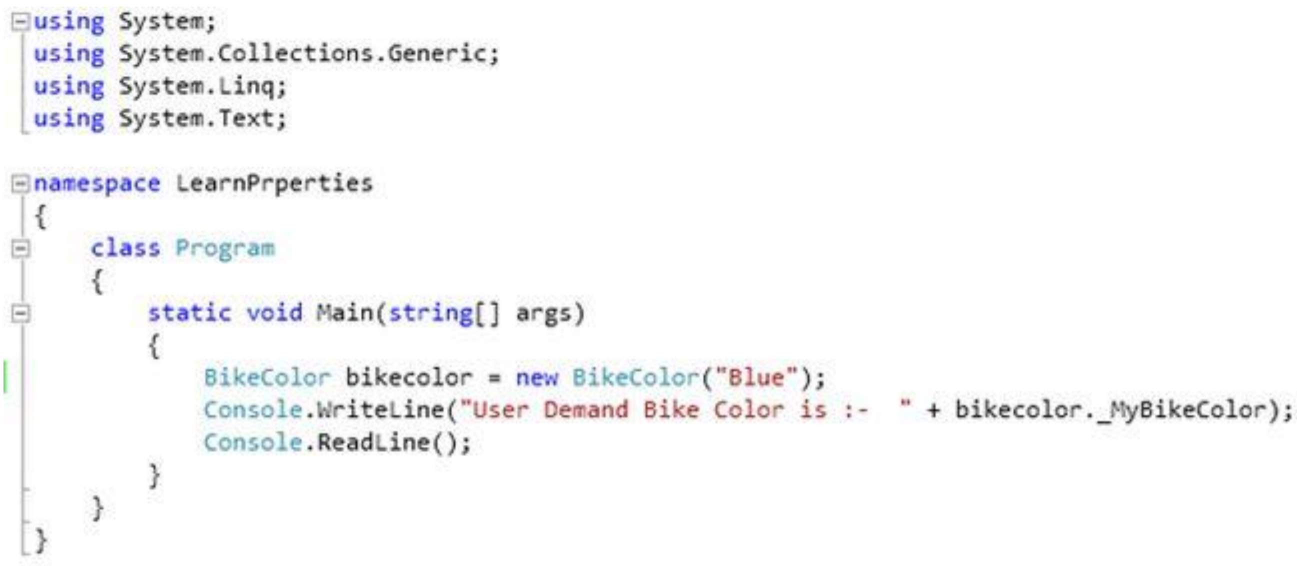
Press F5. It will show the following error.



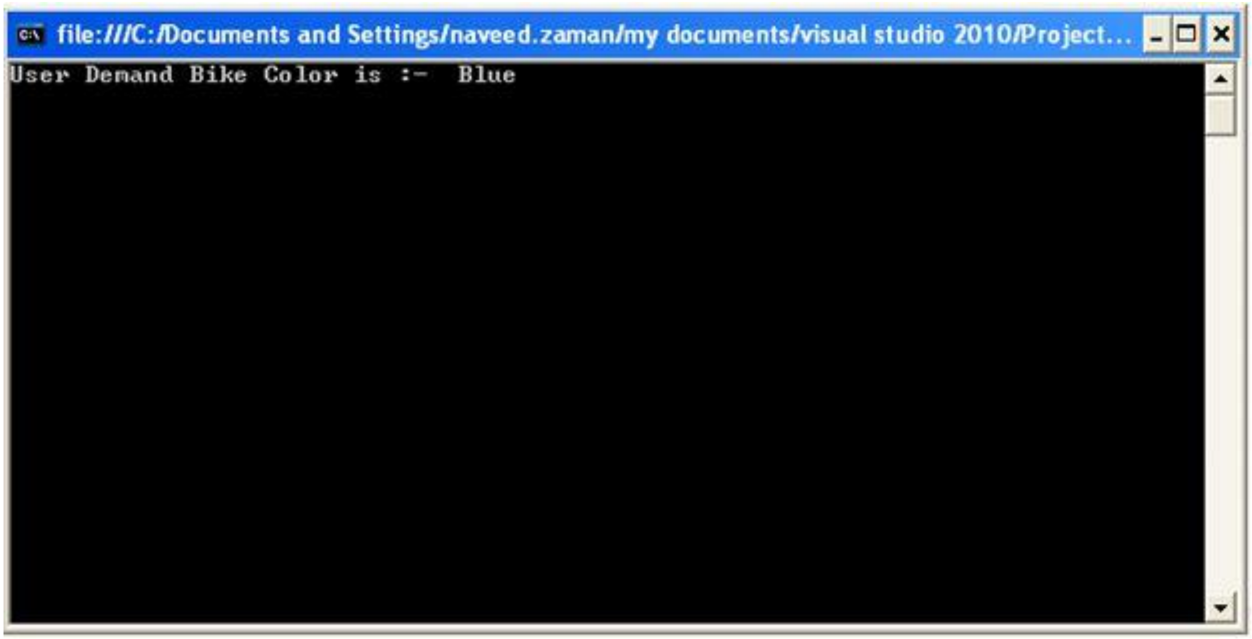
1 'LearnPrperties.BikeColor.MyBikeColor' is inaccessible due to its protection level

Step 6

Now we will try to access the private variable using the property _MyBikeColor. Then it will work fine.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace LearnPrperties
{
    class Program
    {
        static void Main(string[] args)
        {
            BikeColor bikecolor = new BikeColor("Blue");
            Console.WriteLine("User Demand Bike Color is :-  " + bikecolor._MyBikeColor);
            Console.ReadLine();
        }
    }
}
```



file:///C:/Documents and Settings/naveed.zaman/my documents/visual studio 2010/Project...

```
User Demand Bike Color is :-  Blue
```

Conclusion

After creating this simple example we have learned the importance of properties in OOP. Properties help us to access the private variable of the class. Moreover, properties help us to protect the private variable of the class from the unauthorized access.

3. More about C# Class Properties

Types of Properties

The following are the types of properties:

- Read only property.
- Write only property.
- Auto implemented property.

Read only Properties

Using a simple technique we can apply the read only property by just defining the get accessor in the property implementation.

Example

```
using System;

namespace Readonlyproperties
{
    class Program
    {
        static void Main(string[] args)
        {
            Car SportCar = new Car();

            Console.WriteLine(
                "Model: {0}",SportCar.Model_ID);

            Console.WriteLine(
               "Maker: {0}",SportCar.Maker_Name);

            Console.ReadKey();
        }
        public class Car
        {
            private int model = 2012;
            private string maker = "mercedes";

            public int Model_ID
            {
                get
                {
                    return model;
                }
            }

            public string Maker_Name
            {
                get
                {
                    return maker;
                }
            }
        }

    }
}
```

In the example above we create the Car class with two read-only properties, Model_ID and Maker_Name. You can say that each property is read-only because they only have get accessors. We assign the values for the model and maker at the time of defining the variables. In this case, the values are 2012 and "mercedes".

The Main method of the Readonlyproperties class instantiates a new Car object named SportCar. The instantiation of SportCar uses the default constructor of the Car class.

Since the model and maker properties of the Car class are read-only, if you insert SportCar.Model_ID = 2013 into the example, the program will generate an error and not compile, because Model_ID is read-only; the same goes for Maker_Name. When the Model_ID and Maker_Name properties are used in Console.WriteLine, they work fine. This is because these are read operations that only invoke the get accessor of the Model_ID and Maker_Name properties.

Write only Property

We have discussed read only properties. Now we will discuss the write-only property; there is very little difference between the read-only and write-only properties. A write-only property only has a set accessor.

Example

```
using System;

namespace WriteOnlyProperty
{
    class Program
    {
        static void Main(string[] args)
        {
            Car SportsCar = new Car();
            SportsCar._model = 2013;
            SportsCar._maker = "lamborghini";
            SportsCar.DisplayCustomerData();
            Console.ReadKey();

        }
        public class Car
{
    private int model = -1;
    public int _model
    {
        set
        {
            model = value;
        }
    }


    private string maker = string.Empty;

    public string _maker
    {
        set
        {
            maker = value;
        }
    }

    public void DisplayCustomerData()
    {
        Console.WriteLine(
                "Model: {0}", model );

        Console.WriteLine(
           "Maker: {0}", maker );

    }
}

    }

}
```

In the example above we create the Car class with two write-only properties, Model_ID and Maker_Name. You can say that each property is write-only because they only have set accessors. Using the set property we have assigned values to model and maker. In this case, the values are 2013 and "lamborghini".

The Main method of the WriteOnlyProperty class instantiates a new Car object named SportCar. The instantiation of SportCar uses the default constructor of the Car class.

Since the model and maker properties of the Car class are write-only, if you inserted Console.WriteLine (SportCar. Model_ID) into the example, the program will generate an error and not compile, because Model_ID is write-only; the same goes for Maker_Name. When the Model_ID and Maker_Name properties are used in SportCar.Model_ID =2012, they work fine. This is because these are write operations that only invoke the set accessor of the Model_ID and Maker_Name properties.

Auto implemented Property

C# 3.0 introduced a new class property, called auto implemented property, that creates properties without get and set accessor implementations.

Example

```
using System;

namespace AutoImplementedproperty
{
    class Program
    {
        static void Main(string[] args)
        {
            car SportCar = new car();

            SportCar.model = 2014;
            SportCar.maker = "ferrari";

            Console.WriteLine(
                "Model: {0}", SportCar.model);

            Console.WriteLine(
               "Maker: {0}", SportCar.maker);


            Console.ReadKey();

        }
        public class car
        {
            public int model { get; set; }
            public string maker { get; set; }
        }

    }
}
```

Please note that the get and set accessor in this example do not have implementations. In an an auto-implemented property, the C# compiler performs the traditional properties behind the scenes. Main methods use the same traditional properties in auto-implemented property that we discussed earlier.

4. Constructor

Types of Properties

- Default Constructor
- Constructor Overloading
- Private Constructors
- Constructor Chaining
- Static Constructors
- Destructors

Please note a few basic concepts of constructors and ensure that your understanding is crystal clear, otherwise you can't understand OOP (constructors).

1. Constructors have the same name as the class name.
2. The purpose of constructors is for initialization of member variables.
3. A constructor is automatically invoked when the object is created.
4. A constructor doesn't have any return type, not even void.
5. If we want some code to be executed automatically then the code that we want to execute must be put in the constructor.
6. We can't call the constructor explicitly.

The general form of a C# constructor is as follows:

```
modifier constructor_name (parameters)
{
//constructor body
}
```

The modifiers can be private, public, protected or internal. The name of a constructor must be the name of the class for which it is defined. A constructor can take zero or more arguments. A constructor with zero arguments (that is no-argument) is known as the default constructor. Remember that there is not a return type for a constructor.

Default Constructors

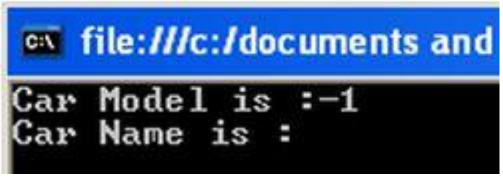
A constructor without arguments is known as the default constructor. Remember that there is no return type for a constructor. That default constructor simply invokes the parameterless constructor of the direct base class.

Example 1

```
using System;
namespace DefaultConstructors
{
   class Program
   {
      static void Main(string[] args)
      {
         car sportscar = new car();
```

```
            Console.WriteLine("Car Model is :{0} ", sportscar.model_Id);
            Console.WriteLine("Car Name is :{0}", sportscar.Maker_Name);
            Console.ReadKey();
        }
        class car
        {
            private int model=-1;
            private string maker = string.Empty;

            public car()
            {
            //Default Constructor
            }
            public int model_Id
            {
                get {
                        return model ;
                    }
            }
            public string Maker_Name
            {
                get
                {
                    return maker;
                }
            }
        }
    }
}
```

Output Example 1



```
Car Model is :-1
Car Name is :
```

In this simple example we have the constructor without arguments or zero parameters that is the default constructor of the class. The output of the preceding example is empty member variables.

Please note one more point; if we remove the following code from Example 1 then the output is the same.

```
public car()
{
   //Default Constructor
}
```

Which means that if we did not define the constructor of the class the system will call the default constructor.

Constructors Overloading

First we will discuss the purpose of constructor overloading; it's very important to have the clear understating of the preceding topic. There are many complex conditions that exist when designing OOP models and we need to initialize a different set of member variables for different purposes of a class. So we need to use constructor overloading. The definition of constructor overloading is:

Just like member functions, constructors can also be overloaded in a class. The overloaded constructor must differ in their number of arguments and/or type of arguments and/or order of arguments.

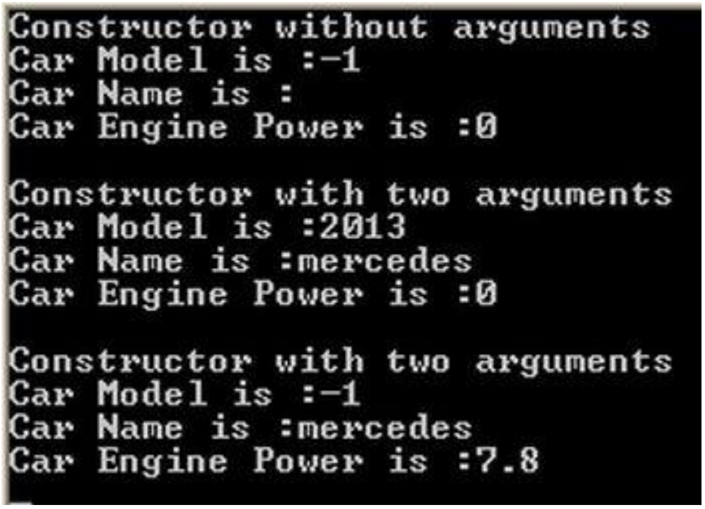
Example 2

```
using System;

namespace ConstructorsOverloading
{
    class Program
    {
        static void Main(string[] args)
        {

            car sportscar1 = new car();
            car sportscar2 = new car(2013, "mercedes");
            car sportscar3 = new car("mercedes", 7.8);

            Console.WriteLine("Constructor without arguments");
            Console.WriteLine("Car Model is :{0} ", sportscar1.model_Id);
            Console.WriteLine("Car Name is :{0}", sportscar1.Maker_Name);
            Console.WriteLine("Car Engine Power is :{0}", sportscar1.Engine);

            Console.WriteLine("\nConstructor with two arguments");
            Console.WriteLine("Car Model is :{0} ", sportscar2.model_Id);
            Console.WriteLine("Car Name is :{0}", sportscar2.Maker_Name);
            Console.WriteLine("Car Engine Power is :{0}", sportscar2.Engine);

            Console.WriteLine("\nConstructor with two arguments");
            Console.WriteLine("Car Model is :{0} ", sportscar3.model_Id );
            Console.WriteLine("Car Name is :{0} ", sportscar3.Maker_Name);
            Console.WriteLine("Car Engine Power is :{0}", sportscar3.Engine );

            Console.ReadKey();
        }
        class car
        {
            private int model = -1;
            private string maker = string.Empty;
            private double  Enginetype= 0.0;
```

```
public car()
{
    //Default Constructor
}
public car( int _model,string _maker)
{
    model = _model;
    maker = _maker;
}
public car(string _maker, double _power)
{
    maker = _maker;
    Enginetype = _power;
}
public int model_Id
{
    get
    {
        return model;
    }
    set
    {
        model = value;
    }
}
public string Maker_Name
{
    get
    {
        return maker;
    }
    set
    {
        maker = value;
    }
}
public double  Engine
{
    get
    {
        return Enginetype;
    }
    set
    {
        Enginetype = value;
    }
}
```

```
        }
    }
}
```

Output Example 2



```
Constructor without arguments
Car Model is :-1
Car Name is :
Car Engine Power is :0

Constructor with two arguments
Car Model is :2013
Car Name is :mercedes
Car Engine Power is :0

Constructor with two arguments
Car Model is :-1
Car Name is :mercedes
Car Engine Power is :7.8
```

Dear reader, please note that in this example we have overload the constructor using three different objects, these are sportscar1, sportscar2 and sportscar3.

We notice that:

- sportscar1 has no arguments (the default constructor is out of topic scope). So that member variable has that default values that was assigned at the time of initialization.

- sportscar2 has two arguments that initializes the member variables model and name with the values 2013 and Mercedes respectively but does not initialize the Enginetype variable so it has the default value zero.

- sportscar3 has two arguments that initialize the member variables name and Enginetype with the values Mercedes and 7.8 respectively but does not initialize the model variable so it is has the default value -1.

Private Constructors

Dear reader, as we know, the private access modifier is a bit of a special case. We neither create the object of the class, nor can we inherit the class with only private constructors. But yes, we can have the set of public constructors along with the private constructors in the class and the public constructors can access the private constructors from within the class through constructor chaining. Private constructors are commonly used in classes that contain only static members.

```
using System;
namespace PrivateConstructors
{
    class Program
    {
        static void Main(string[] args)
        {
            car sportscar = new car(2013);
            Console.Read();
        }
        class car
        {
            public  string  carname;
            private car()
            {
              carname = "lamborghini";
            }
            public car(int model):this()
            {
                Console.WriteLine("Model Year:{0}",model);
                Console.WriteLine("Maker Name:{0}",carname);
            }

        }
    }
}
```

This is a very simple example of private constructors in which we use a public constructor to call the private constructor.

Constructors Chaining

Dear reader, when a constructor invokes another constructor in the same class or in the base class of this class it is known as constructor chaining. It is a very useful technique for avoiding code duplication.

```
using System;
namespace Constructorchaining
{
    class Program
    {
        static void Main(string[] args)
        {
            car sportscar = new car(7.8);
            Console.Read();
        }
        class car
        {
            public string carname;
            public int model;
            public double engine;
            public car(string _carname)
            {
                carname = _carname;
            }

            public car(int _model): this("lamborghini")
            {
                model= _model;

            }
            public car(double _engine) : this(2013)
            {
                engine = _engine;
                Console.WriteLine("Model Year:{0}", model);
                Console.WriteLine("Maker Name:{0}", carname);
                Console.WriteLine("Engine Type:{0}", engine);
            }

        }
    }
}
```

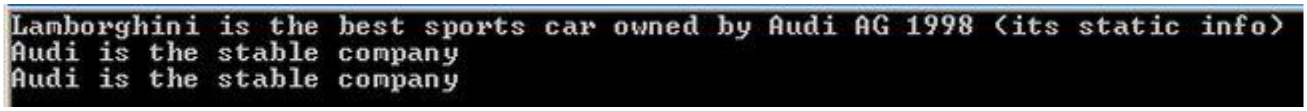
In the preceding example we created three different classes with the name car having different parameter types, each one chain the previous one to invoke the other constructor.

Static Constructors

The static constructor is the special type that does not take access modifiers or have parameters. It is called automatically to initialize the class before the first instance is created or any static members are referenced. The static constructor is not called directly. Users can't control the execution of the static constructor.

```
using System;
namespace staticconstructors
{
    class Program
    {
        public class car
        {
            static car()
            {
                System.Console.WriteLine(@"Lamborghini is the best sports car owned by Audi AG 1998 (its static info)");
            }
            public static void Drive()
            {
                System.Console.WriteLine("Audi is the stable company");
            }
        }
        static void Main()
        {
            car.Drive();
            car.Drive();
            Console.ReadKey();
        }
    }
}
```

Output



```
Lamborghini is the best sports car owned by Audi AG 1998 (its static info)
Audi is the stable company
Audi is the stable company
```

Note that the static constructor is called when the class is loaded the first time. When we again call "car.Drive" it will not call the static constructor.

Destructors

Dear reader, when we are working with OOP and creating objects in system memory it's necessary to clean the unwanted objects from the system memory The .NET framework has built-in Garbage Collection to de-allocate memory occupied by the unused objects. A Destructor is a function with the same name as the name of the class but starting with the character ~. The Destructor can't have any of the modifiers private, public etcetera.

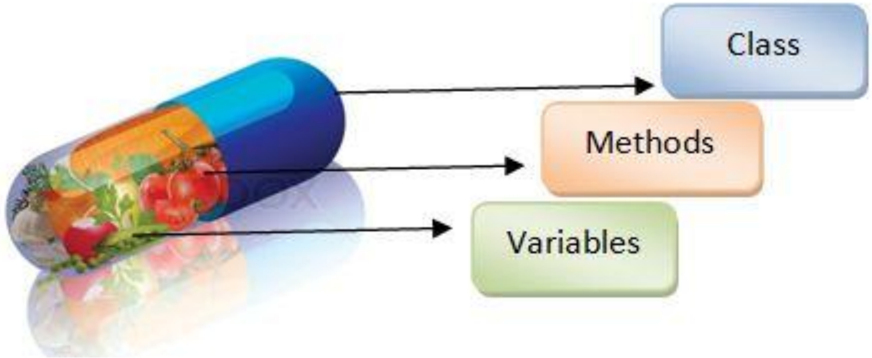
Example

```
class car
{
public car()
{
// constructor
}
~car()
{
// Destructor
}
}
```

5. Encapsulation

"*Encapsulation"* is a process of binding data members (variables and properties) and member functions (methods) into a single unit". And Class is the best example of encapsulation.

Here is a very simple diagram that will explain encapsulation itself:



Key Points of Encapsulation

- Protection of data from accidental corruption
- Specification of the accessibility of each of the members of a class to the code outside the class
- Flexibility and extensibility of the code and reduction in complexity
- Encapsulation of a class can hide the internal details of how an object does something
- Using encapsulation, a class can change the internal implementation without affecting the overall functionality of the system
- Encapsulation protects abstraction

In very simple words we can say that encapsulation is a technique to hide the complexity of a class (its member variables and methods) from the user. It makes it easier for the user to use only the accessible required methods plus it will protect the data from accidental corruption.

Example

```
using System;
namespace Encapsulation
{
    class Program
    {
        static void Main(string[] args)
        {
            credit_Card_Info cr = new credit_Card_Info("Naveed Zaman", "002020-1",5000,16.5);
            cr.disbursement=6999;
            cr.display();
            Console.ReadKey();
        }
        class credit_Card_Info
        {
            private string customername=string.Empty  ;
            private string cardno = string.Empty;
            private double creditamount=-1;
            private double markuprate = -1;

            public credit_Card_Info(string _customername, string _cardno, double _creditamount, double _markuprate)
            {
                customername = _customername;
                cardno = _cardno;
                creditamount = _creditamount;
                markuprate = _markuprate;
            }

            public double disbursement
            {
                get
                {
                    return creditamount;
                }
                set
                {
                    creditamount = creditamount + value;
                    creditamount = creditamount * markuprate / 100 + creditamount;
                }

            }

             public void display()
            {
                Console.WriteLine("Customer Name :{0}", customername);
                Console.WriteLine("Card Number :{0}", cardno);
                Console.WriteLine("Markup Rate :{0}", markuprate);
                Console.WriteLine("Current Balance with Markup:{0}", creditamount);

            }


        }
    }
}
```

In that example we have created the class credit_Card_Info with four member variables having private access modifiers. Then we have defined the constructor of the class that initializes the four member variables of a class. After that we used get and set properties of the class to access the private variable of a class. The last display method will show the output of the class member variables.

So we can observe that the user must provide only the "cr.disbursement=6999" amount and encapsulation will calculate the markup and outstanding of the client. In that simple example the user did know the complexity of the class member variables and methods.

Another very common example of encapsulation is TextBox.

For Example:

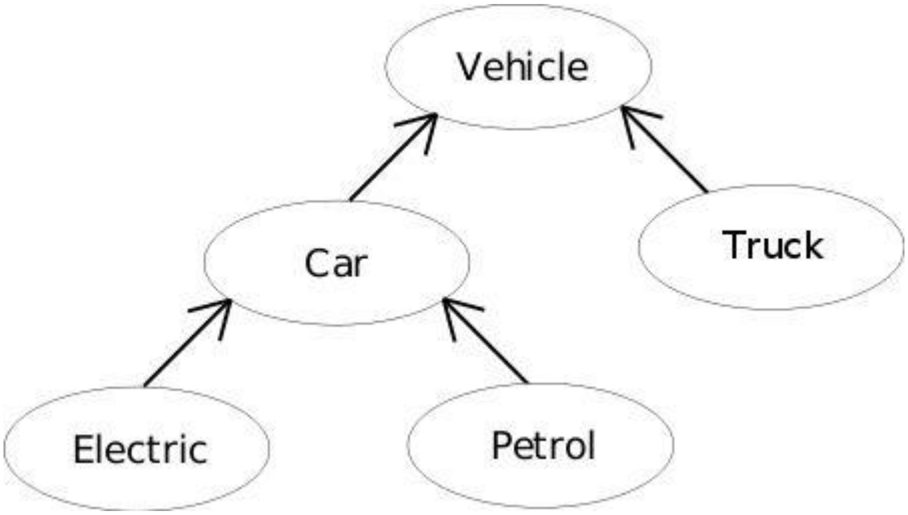
```
TextBox tb = new TextBox();
tb.Text ="Hello World";
```

6. Inheritance

Inheritance means to inherit something from the source. For example a son can inherit habits from his father. The same concept is used in of Object Oriented Programming; it's the second pillar of OOP.

Inheritance enables creation of a new class that inherits the properties from another class or base class and the class that inherits those members is called the derived class. So the derived class has the properties of the base class and its own class properties as well.

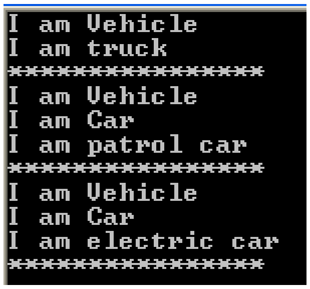
Here is a very simple diagram that will explain inheritance:



Example:

```
using System;
namespace Inheritance
{
    class Program
   {
      public class vehicle
      {
         public vehicle()
         {
            Console.WriteLine("I am Vehicle");
         }
      }

      public class car : vehicle
      {
         public car()
         {
            Console.WriteLine("I am Car");
            }
        }

        public class truck : vehicle
        {
            public truck()
            {
                Console.WriteLine("I am truck");
            }
        }

        public class electric : car
        {
            public electric()
            {
                Console.WriteLine("I am electric car");
            }
        }

        public class petrol : car
        {
            public petrol()
            {
                Console.WriteLine("I am patrol car");
            }
        }

        static void Main(string[] args)
        {
            truck tr = new truck();
            Console.WriteLine("****************");
            petrol pr = new petrol();
            Console.WriteLine("****************");
            electric el = new electric();
            Console.WriteLine("****************");
            Console.ReadKey();
        }
    }
}
```

Out Put:



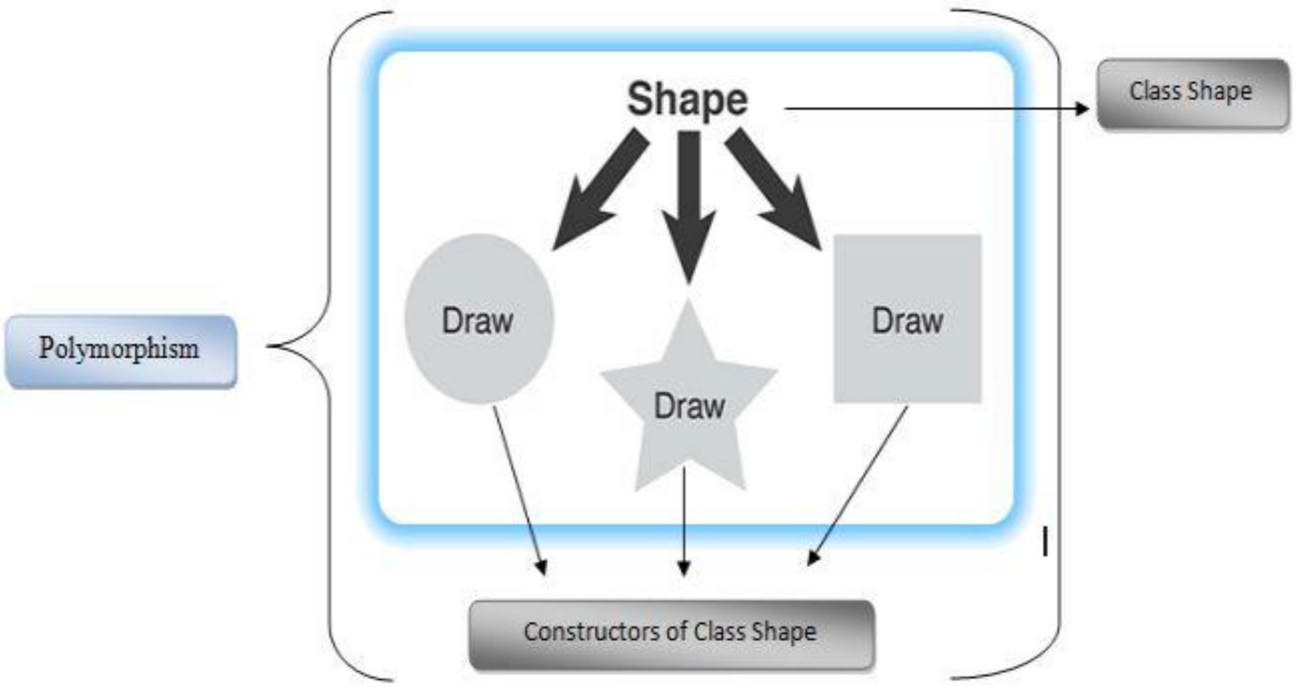
```
I am Vehicle
I am truck
****************
I am Vehicle
I am Car
I am patrol car
****************
I am Vehicle
I am Car
I am electric car
****************
```

In this simple example we have designed a base class, vehicle, and then we derived two classes, car and truck; these are derived classes of the vehicle class. After that we again create two classes derived from the car class. The Patrol and electric classes are derived classes of the car class which is the base class for the derived classes. Then we just create the object of the truck class that automatically calls the base class vehicle and the same for the patrol and electric classes, we create an object of these classes that automatically call the car class and then the car class calls the vehicle class.

7. Polymorphism

How often do we use polymorphism in our programs? Polymorphism is the third main pillar of Object Oriented Programming languages and we use it nearly daily without thinking about it.

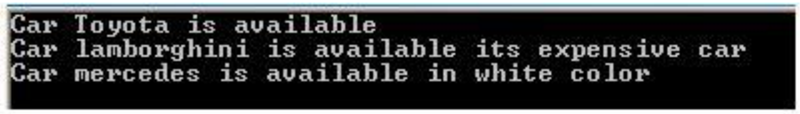
Here is a very simple diagram that will explain the polymorphism itself.



In simple words we can say that whenever we are overloading the methods of a class it is called polymorphism. Or you can say polymorphism is often expressed as "one interface, multiple functions". This means we have more than one function with the same name but different parameters.

Example

```
using System;
namespace Polymorphism
{
    class Program
    {
        class car
        {
            public  void CarDetail()
            {
                Console.WriteLine("Car Toyota is available");
            }

            public void CarDetail(int priceRange)
            {
                Console.WriteLine("Car lamborghini is available its expensive car");
            }

            public void CarDetail(int priceRange, string colour)
            {
                Console.WriteLine("Car mercedes is available in white color");
            }
        }
        static void Main(string[] args)
        {
            car cr = new car();
            cr.CarDetail();
            cr.CarDetail(2200000);
            cr.CarDetail(2200000, "White");
            Console.ReadKey();
        }
    }
}
```



```
Car Toyota is available
Car lamborghini is available its expensive car
Car mercedes is available in white color
```

In this example we have created three different functions with the same name (CarDetail) having a different set of parameters. In the next topic I will discuss Polymorphism in more detail with its two types:

1. Static Polymorphism
2. Dynamic Polymorphism

8. Abstract Methods

"Abstraction is used to manage complexity. No objects of an abstract class are can be created. An abstract class is used for inheritance."

For Example

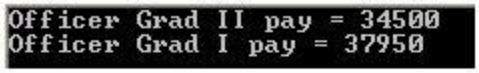
When we drive a car we often need to change the gears of the vehicle but we are otherwise not concerned about the inner details of the vehicle engine. What matters to us is that we must shift gears, that's it. This is abstraction; show only the details that matter to the user.

Example

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace @abstract
{
    class Program
    {

    abstract class pay   // Abstract class
    {
        protected int _basicpay = 20000;
        protected int _houserent = 15000;
        protected int _Tax = -500;
        protected int _NetPay = -500;

        public abstract int gradtwo    { get; }
        public abstract int gradone { get; }

    }

            class Netpay : pay
            {
                public  void CalculatePay()
                {
                    _NetPay =_basicpay + _houserent + _Tax;
                }

                public override int gradtwo   // overriding property
                {
                    get
                    {
                        return _NetPay;
                    }
                }

                public override int gradone   // overriding property
                {
                    get
                    {
                        return _NetPay = _NetPay + _NetPay * 10 / 100;
                    }
                }
            }
        static void Main(string[] args)
        {
             Netpay o = new Netpay();
            o.CalculatePay();
            Console.WriteLine("Officer Grad II pay = {0} \nOfficer Grad I pay = {1}", o.gradtwo, o.gradone);
            Console.ReadKey();
        }
    }
}
```

Output



```
Officer Grad II pay = 34500
Officer Grad I pay = 37950
```

Dear reader, I need your extra concentration for this.

Step 1

```
abstract class pay   // Abstract class
{
    protected int _basicpay = 20000;
    protected int _houserent = 15000;
    protected int _Tax = -500;
    protected int _NetPay = -500;
    public abstract int gradtwo    { get; }
    public abstract int gradone { get; }
 }
```

I have defined one **abstract** class "pay" with protected variable that can only be accessed by the same class or in a derived class. These member variables are initiated with values.

Step 2

```
    class Netpay : pay
    {
        public void CalculatePay()
        {
            _NetPay = _basicpay + _houserent + _Tax;
        }

        public override int gradtwo   // overriding property
        {
            get
            {
                return _NetPay;
            }
        }

        public override int gradone   // overriding property
        {
            get
            {
                return _NetPay = _NetPay + _NetPay * 10 / 100;
            }
        }
    }
```

In this step we have defined the class "Netpay" derived from the abstract base class "pay".

In that class we have defined the "CalculatePay" method having public access modifiers to calculate the pay of the employee. During the pay calculation we used a protected variable from the base class. Here we have overriden the two properties "gradone" and "gradtwo" that will return the values of "_NetPay".

Step 3

```
static void Main(string[] args)
{
    Netpay o = new Netpay();
    o.CalculatePay();
    Console.WriteLine("Officer Grad II pay = {0} \nOfficer Grad I pay = {1}", o.gradtwo, o.gradone);
    Console.ReadKey();
 }
```

In the void main session we have created the object of the "Netpay" class. Using the object we call the "CalculatePay" method that will do the calculation of the pay.

So the user is only concerned with the pay of the employee and its output. How this pay is calculated is not necessary to be understood.

9. User Activity Log Using OOP

Key Concept behind the application

Now, we will create a small application. Please follow this link user-activity-log-using-C-Sharp-with-sql-server/ before continuing. Here we will again redesign the same application using the following concepts. It will help us to understand these concepts more clearly plus how to implement them in our own applications.

1. Class
2. Member variables
3. Read only Properties
4. Methods
5. Object
6. Default Constructor
7. Constructor Overloading

Application Detail

Keep track of user activities in a Database Managements System. Specially while working on the Client server environment where we need to keep track of the System IP/System Name, Login Id, Time stamp and the action performed on any of database applications. Either the user edits any client record or does a transition etcetera.

Step 1

Create the table with the name User_Activity_Log.

```
CREATE TABLE [dbo].[User_Activity_Log](
	[UAL_User_Id] [varchar](20) NOT NULL,
	[UAL_Timestamp] [datetime] NOT NULL,
	[UAL_Function_Performed] [nvarchar](100) NOT NULL,
	[UAL_Other_Information] [nvarchar](100) NULL,
	[UAL_IP_Address] [nvarchar](15) NOT NULL,
PRIMARY KEY CLUSTERED
(
	[UAL_User_Id] ASC,
	[UAL_Timestamp] ASC
)WITH (PAD_INDEX  = OFF, STATISTICS_NORECOMPUTE  = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS  =ON, AL
LOW_PAGE_LOCKS  = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
```

Step 2

Create a Login table.

```
CREATE TABLE [dbo].Login_Client(
	[LON_User_Name] [varchar](20) NULL,
	[LON_Login_Name] [varchar](20) NULL,
	[LON_Employee_No] [varchar](10) NULL,
	[LON_Login_Password] [varchar](20) NULL,
	[LON_Type] [nvarchar](20) NULL,
	[LON_Status] [varchar](20) NULL
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
```

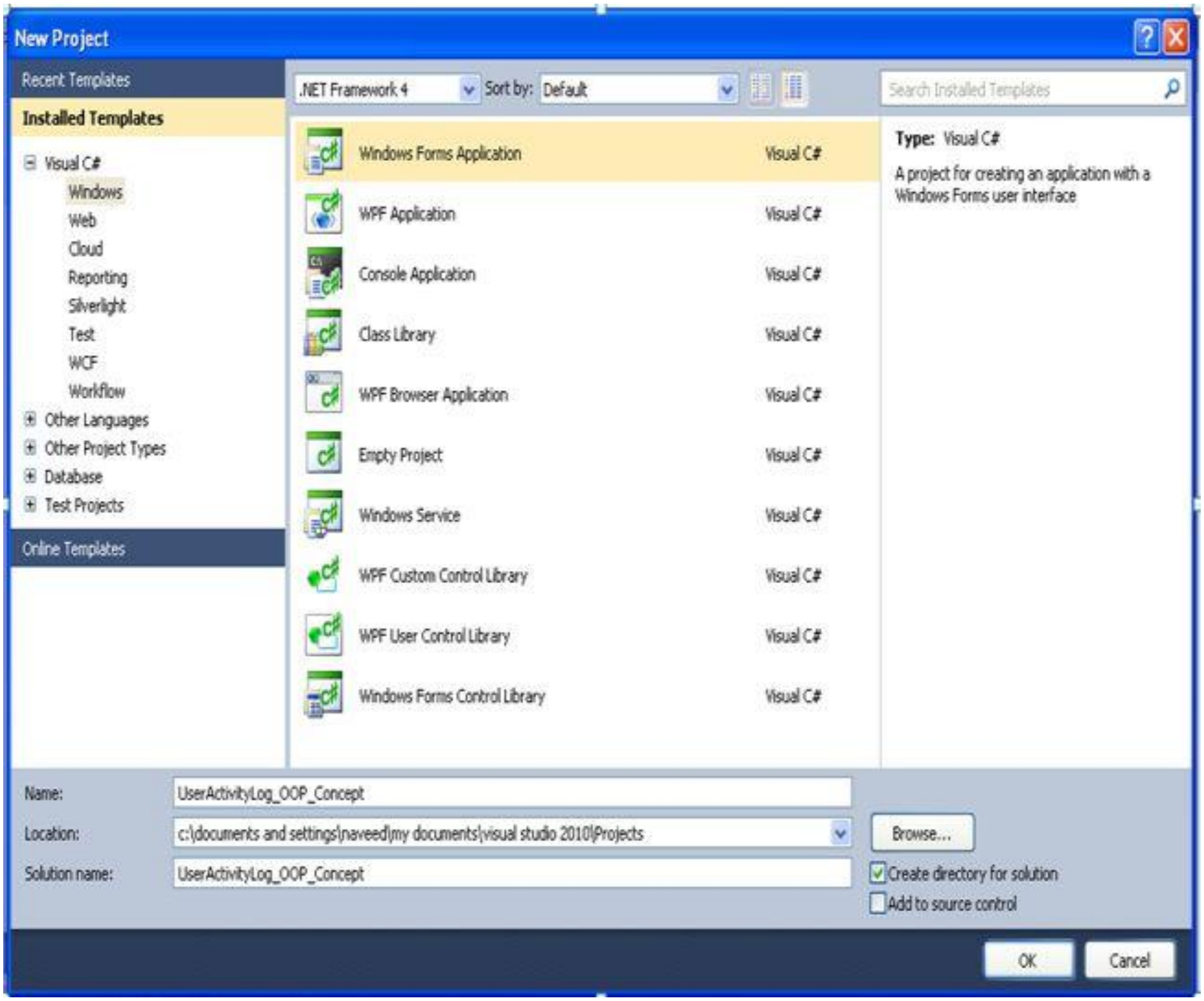
Step 3

Now insert one record in the login table.

```
INSERT INTO PSH.[dbo].[Login_Client]VALUES('Naveed Zaman','naveed.zaman','22339','khan@123','A','Active')
```

Step 4

Start Visual Studio and create a new Desktop project with the name "UserActivityLog_OOP_Concept".


New Project
Recent Templates
Installed Templates
Visual C#
Windows
Web
Cloud
Reporting
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Database
Test Projects
Online Templates
Search Installed Templates
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Windows Service Visual C#
WPF Custom Control Library Visual C#
WPF User Control Library Visual C#
Windows Forms Control Library Visual C#
Browse...
OK
Cancel

.NET Framework 4 Sort by: Default

Type: Visual C#

A project for creating an application with a Windows Forms user interface

Name: UserActivityLog_OOP_Concept

Location: c:\documents and settings\naveed\my documents\visual studio 2010\Projects

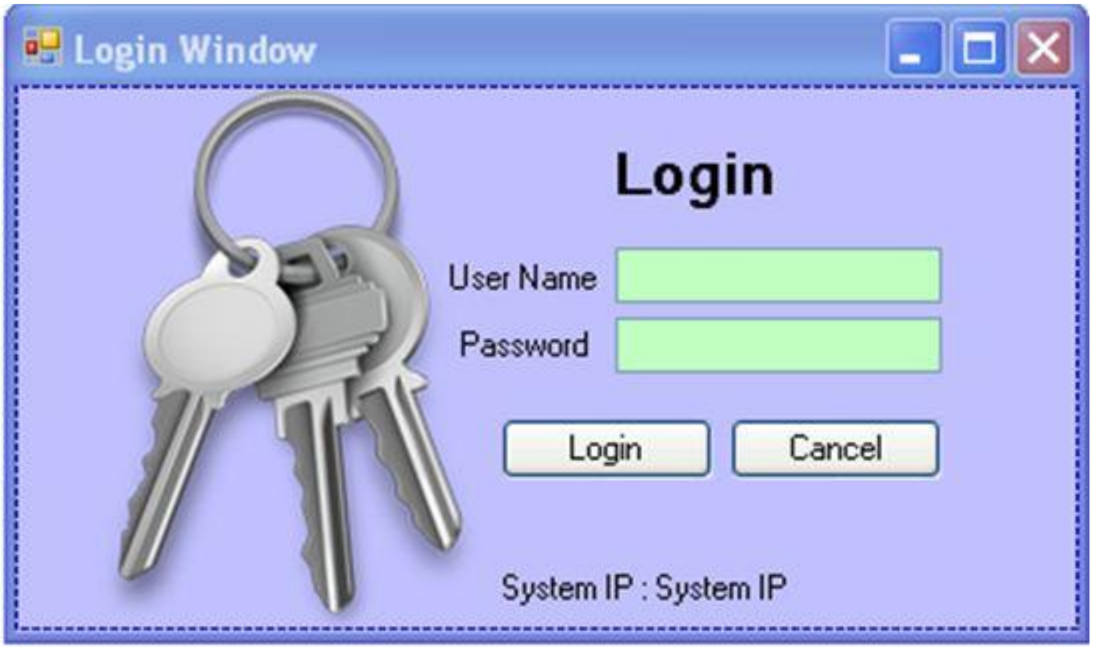
Solution name: UserActivityLog_OOP_Concept

☑ Create directory for solution

□ Add to source control

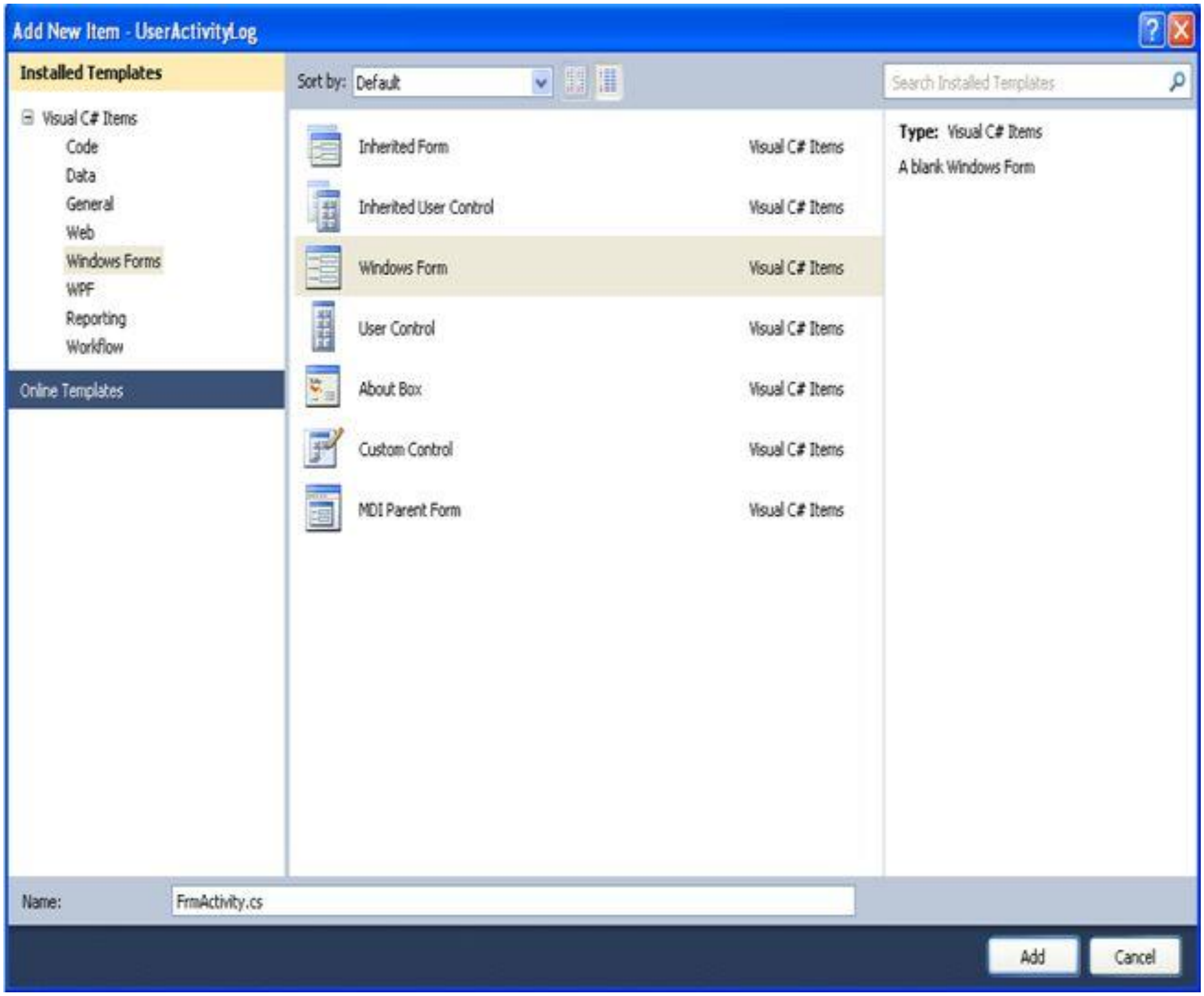
Step 5

First of all we create the login form that will help us to understand the basic concepts of the topic. Create the form as shown in the picture.



Step 6

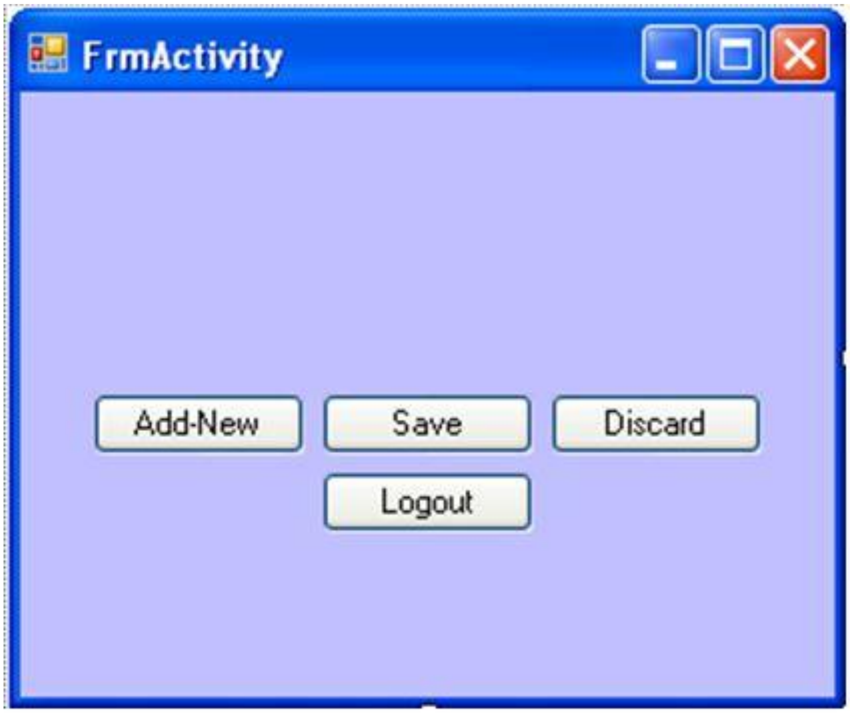
Create another form with the name of frmActivity.



Name: FrmActivity.cs

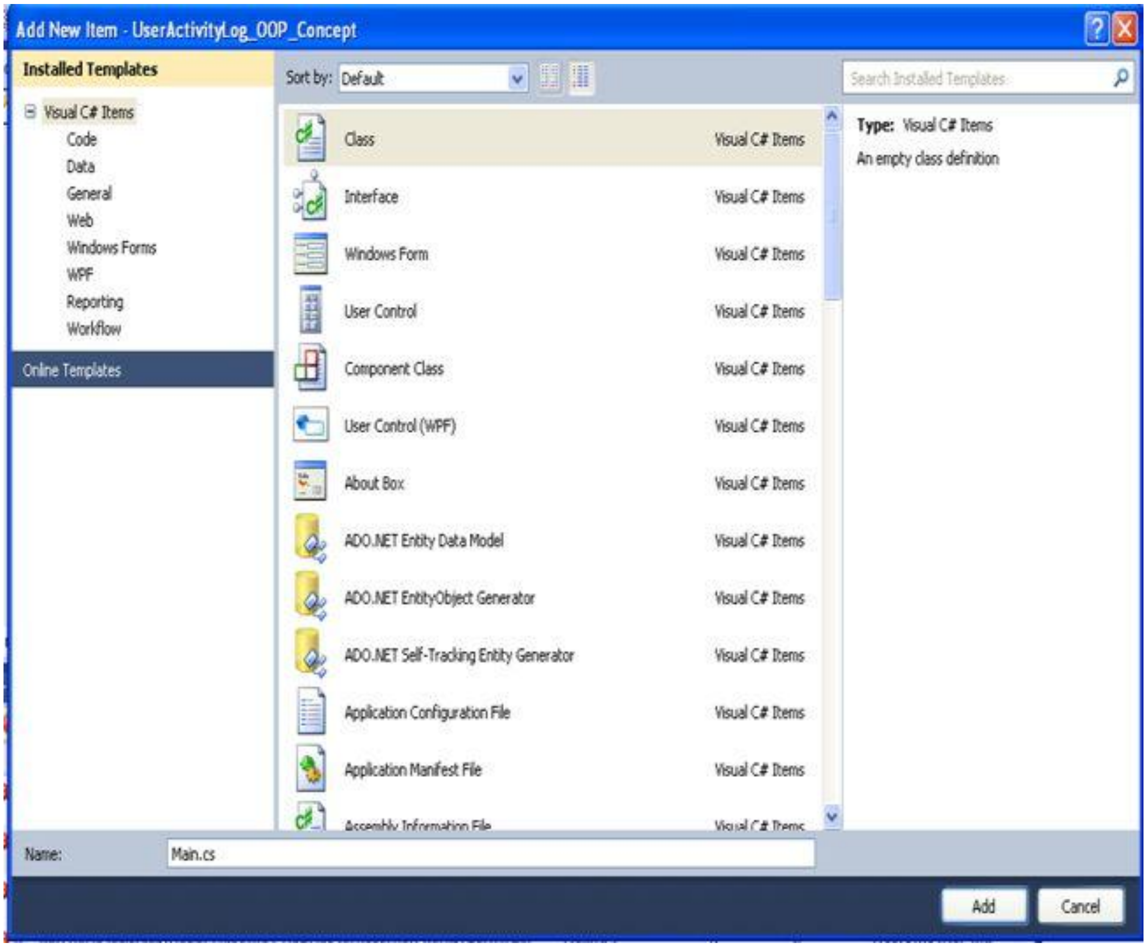
Step 7

Now add a few buttons on it. For example "Add-New", "Save", "Discard" and "Logout" buttons on a form as showm in the following picture.



Step 8

Add a new class with the name "Main".


Add New Item - UserActivityLog_OOP_Concept
Installed Templates
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
Reporting
Workflow
Online Templates
Search Installed Templates
Class Visual C# Items
Interface Visual C# Items
Windows Form Visual C# Items
User Control Visual C# Items
Component Class Visual C# Items
User Control (WPF) Visual C# Items
About Box Visual C# Items
ADO.NET Entity Data Model Visual C# Items
ADO.NET EntityObject Generator Visual C# Items
ADO.NET Self-Tracking Entity Generator Visual C# Items
Application Configuration File Visual C# Items
Application Manifest File Visual C# Items
Assembly Information File Visual C# Items
Add
Cancel

Sort by: Default

Type: Visual C# Items

An empty class definition

Name: Main.cs

Step 9

Now we will create the function GetDBConnection. It is a public function so it can be accessed from anywhere in the project. But you must modify the data source settings relevant to your own settings.
For example:

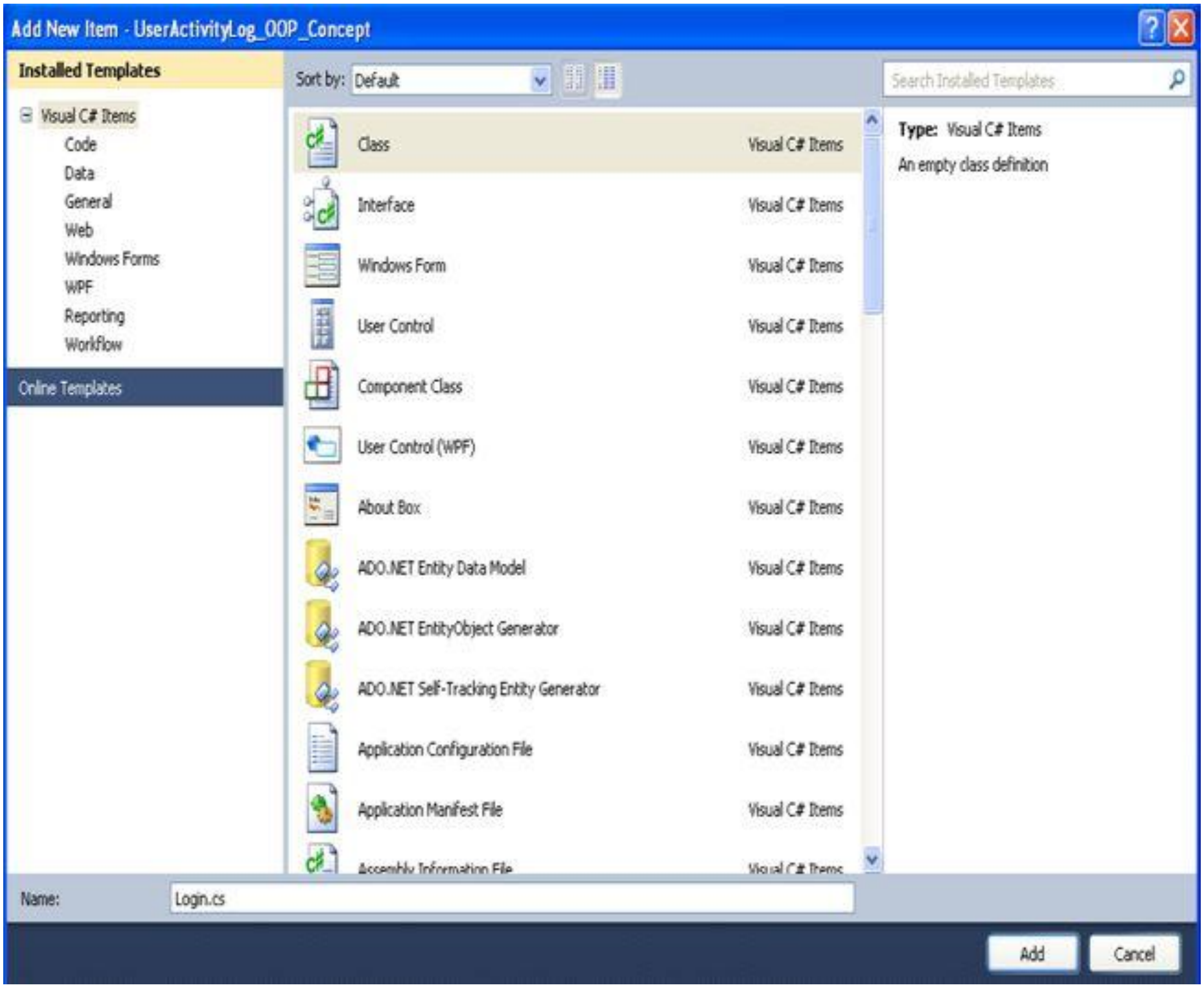
1. SQL Server instance name. (.\\CRMIS)
2. SQL Server user name and password. (user id sa password.###Reno123)
3. Initial Catalog=PSH (database name)

Add the public static class to "Main.cs":

```
public static SqlConnection GetDBConnection()
{
  SqlConnection conn = new SqlConnection(
  "Data Source=.\\CRMIS;Initial Catalog=PSH;User ID=sa;Password=###Reno321");
   return conn;
}
```

Step 10

Now we will create another class login that will help us to verify the anthorication of the user. After providing username and password.


Add New Item - UserActivityLog_OOP_Concept
Installed Templates
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
Reporting
Workflow
Online Templates
Sort by: Default
Search Installed Templates
Class Visual C# Items
Interface Visual C# Items
Windows Form Visual C# Items
User Control Visual C# Items
Component Class Visual C# Items
User Control (WPF) Visual C# Items
About Box Visual C# Items
ADO.NET Entity Data Model Visual C# Items
ADO.NET EntityObject Generator Visual C# Items
ADO.NET Self-Tracking Entity Generator Visual C# Items
Application Configuration File Visual C# Items
Application Manifest File Visual C# Items
Assembly Information File Visual C# Items
Type: Visual C# Items
An empty class definition
Add
Cancel

Name: Login.cs

Step 11

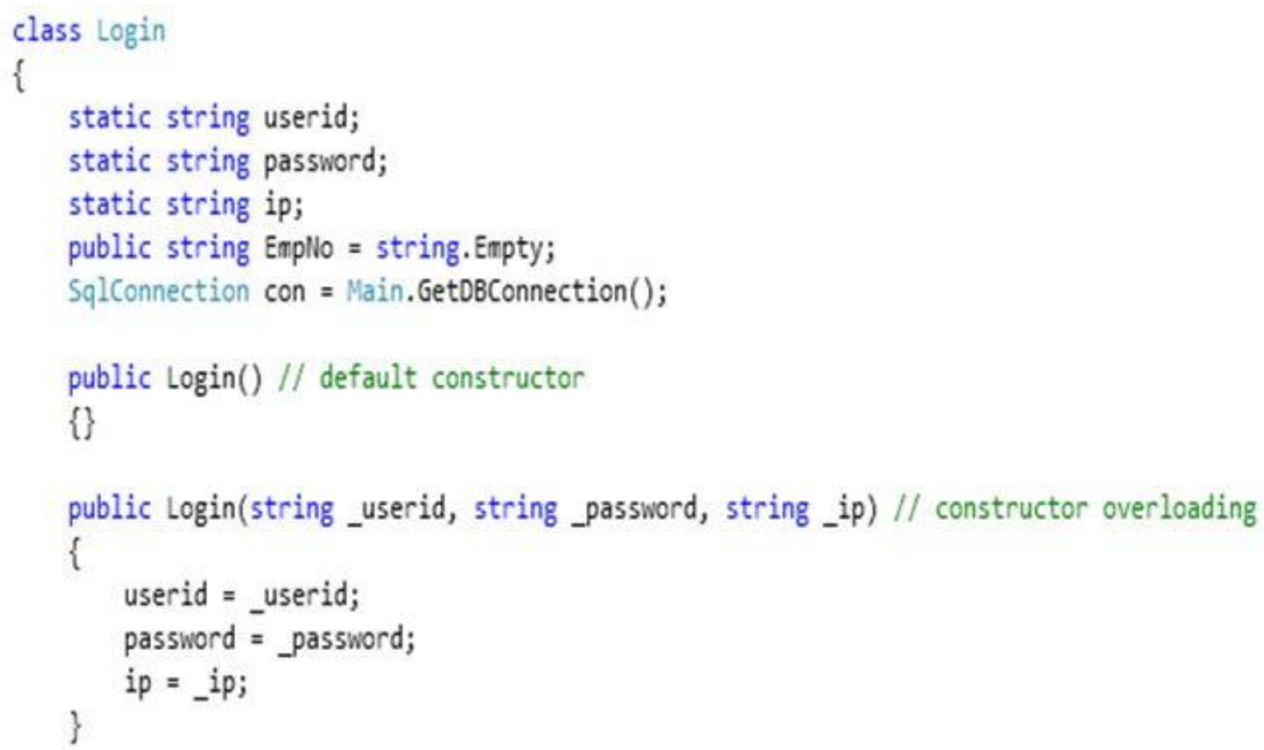
Insert the following code in the class login:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace UserActivityLog_OOP_Concept
{
    class Login
    {
```

```
static string userid;
static string password;
static string ip;
public string EmpNo = string.Empty;
SqlConnection con = Main.GetDBConnection();

public Login()
{}

public Login(string _userid, string _password, string _ip)
{
    userid = _userid;
    password = _password;
    ip = _ip;
}

    public string getidinfo
    {
        get

        {
            return userid;
        }
    }

    public string getpassinfo
    {
        get
        {
            return password;
        }
    }
    public string getipinfo
    {
        get
        {
            return ip;
        }
    }
public  string Validation()
{
    try
    {
        DataTable consultanttable = new DataTable();
        string SQL = @"SELECT LON_Employee_No FROM Login_Client where
                LON_Login_Name ='" + userid + "' AND LON_Login_Password ='" + password + "'";
        SqlDataAdapter Consultantdataadapter = new SqlDataAdapter(SQL, con);
        Consultantdataadapter.Fill(consultanttable);
        foreach (DataRow myrow in consultanttable.Rows)
```

```
                {
                    EmpNo = (myrow[0].ToString());
                }
            }
            catch (InvalidCastException e)
            {
                throw (e);    // Rethrowing exception e
            }
            return EmpNo;
        }

    }
}
```

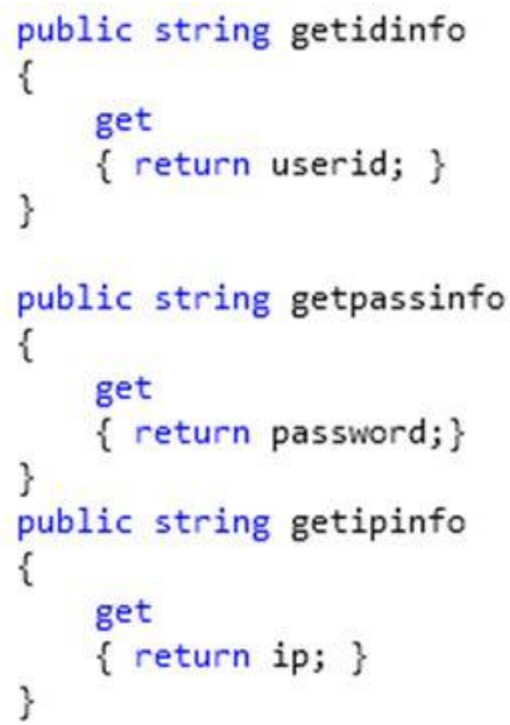
Please check the code carefully.



```
class Login
{
    static string userid;
    static string password;
    static string ip;
    public string EmpNo = string.Empty;
    SqlConnection con = Main.GetDBConnection();

    public Login() // default constructor
    {}

    public Login(string _userid, string _password, string _ip) // constructor overloading
    {
        userid = _userid;
        password = _password;
        ip = _ip;
    }
```

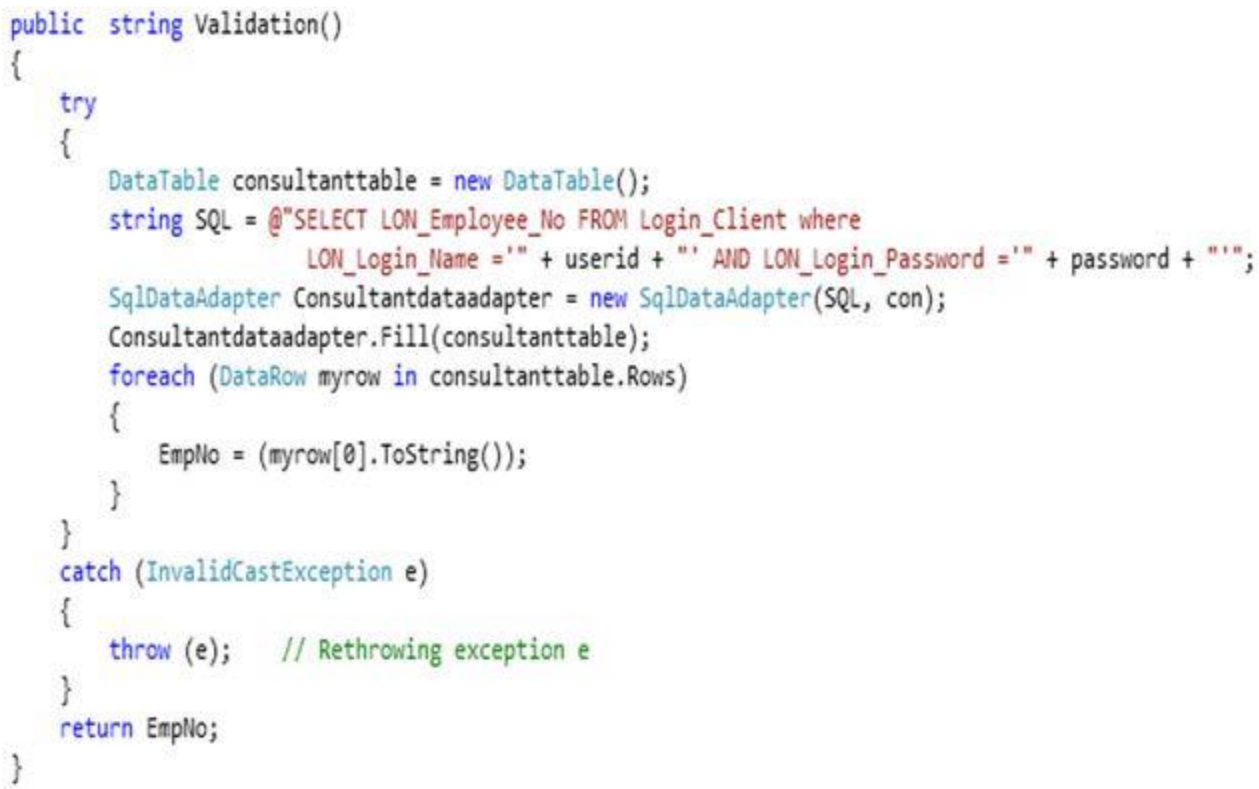
We have create the class login and in that class we define four member variables, three of them static and one is public. We have already discussed access modifiers in prevoius topics. After that we have created an object icon of the class Main that we defined in Step 6. It will help us to create the connnection between C# and SQL Server.

After that we have defined the default constructor of a class Login and overloaded the Constructor with two parameters. Its very important to understand the importance of the overloaded constructor. It will be initlized whenever we create the object of the class with the two parameters, userid and password. We have defined a default constructor as well that will help us when we need an object on a class without parameters.



```
public string getidinfo
{
    get
    { return userid; }
}

public string getpassinfo
{
    get
    { return password;}
}
public string getipinfo
{
    get
    { return ip; }
}
```

Further we have defined read-only properties of the class login that will help us to read member variables.

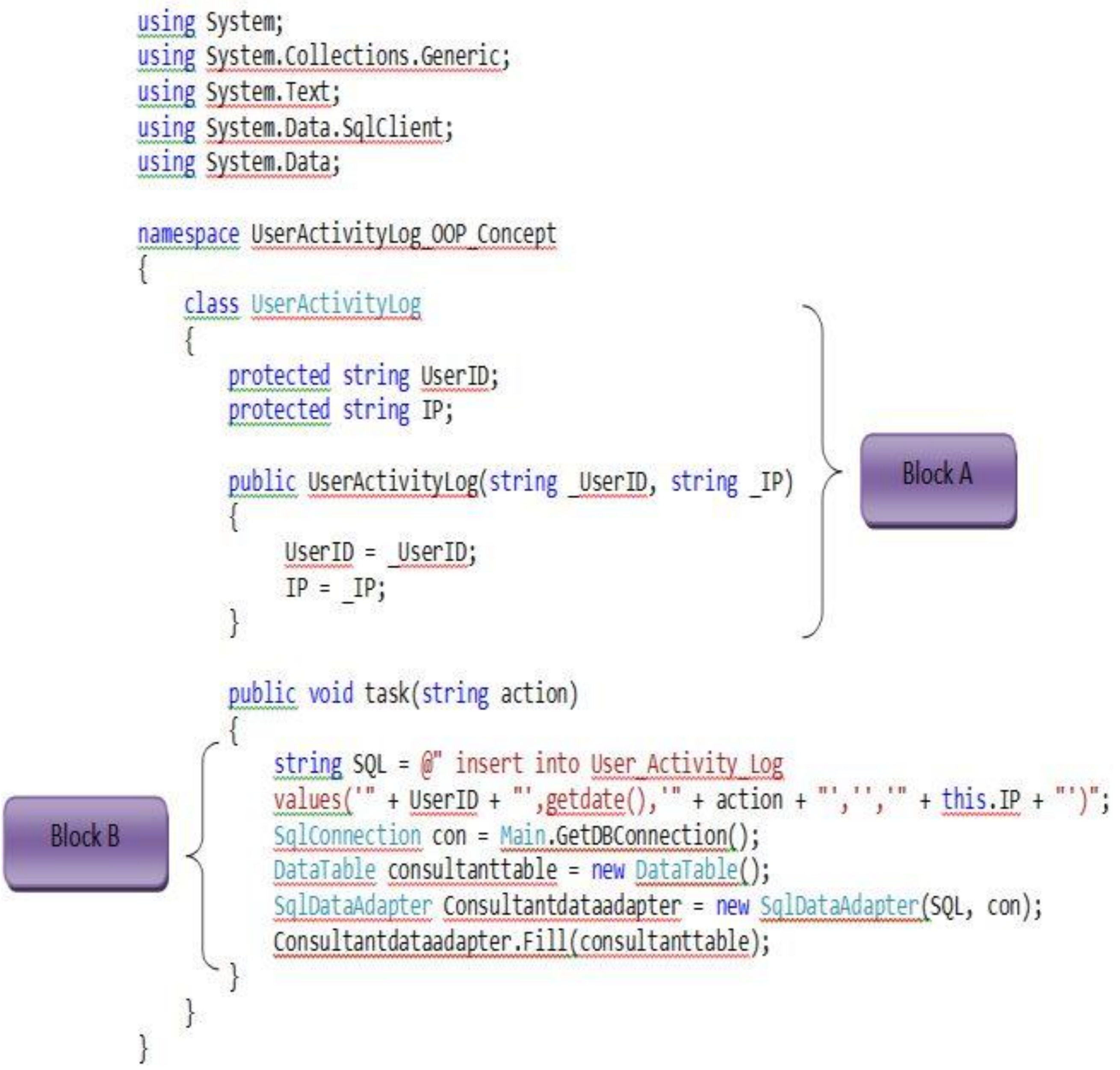


```
public  string Validation()
{
    try
    {
        DataTable consultanttable = new DataTable();
        string SQL = @"SELECT LON_Employee_No FROM Login_Client where
                    LON_Login_Name ='" + userid + "' AND LON_Login_Password ='" + password + "'";
        SqlDataAdapter Consultantdataadapter = new SqlDataAdapter(SQL, con);
        Consultantdataadapter.Fill(consultanttable);
        foreach (DataRow myrow in consultanttable.Rows)
        {
            EmpNo = (myrow[0].ToString());
        }
    }
    catch (InvalidCastException e)
    {
        throw (e);    // Rethrowing exception e
    }
    return EmpNo;
}
```

In this session we have defined a method validation of a class login that will help us to validate the login info and return EmpNo.

Step 12

Please insert the following code in the UserActivityLog class.



```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace UserActivityLog_OOP_Concept
{
    class UserActivityLog
    {
        protected string UserID;
        protected string IP;

        public UserActivityLog(string _UserID, string _IP)
        {
            UserID = _UserID;
            IP = _IP;
        }

        public void task(string action)
        {
            string SQL = @" insert into User_Activity_Log
            values('" + UserID + "',getdate(),'" + action + "','','" + this.IP + "')";
            SqlConnection con = Main.GetDBConnection();
            DataTable consultanttable = new DataTable();
            SqlDataAdapter Consultantdataadapter = new SqlDataAdapter(SQL, con);
            Consultantdataadapter.Fill(consultanttable);
        }
    }
}
```

Block A

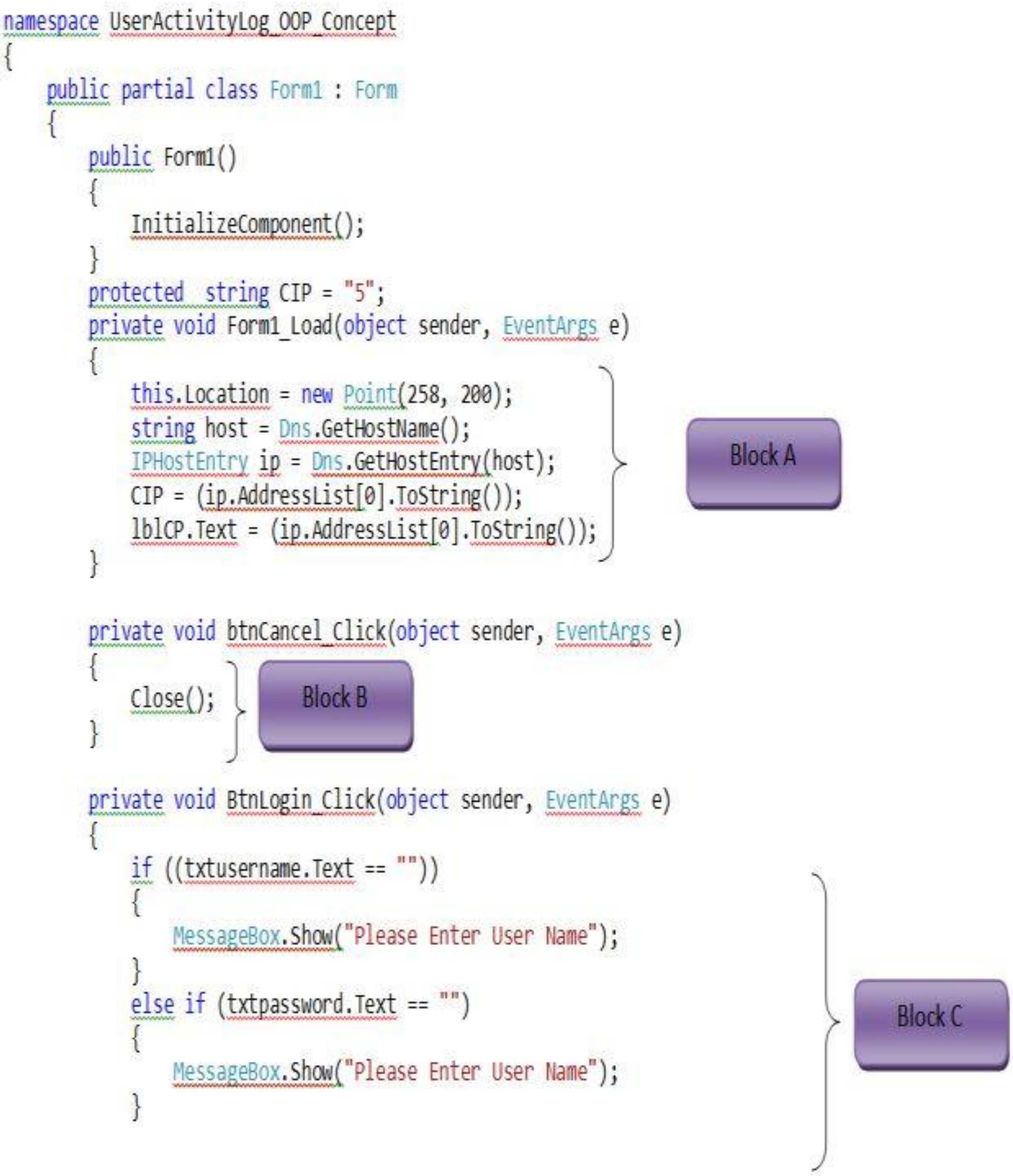
- We have defined two member variables.
- We have defined the constructor of the class User_Activity_Log with two parameter that will initlize the member variables of the class.

Block B

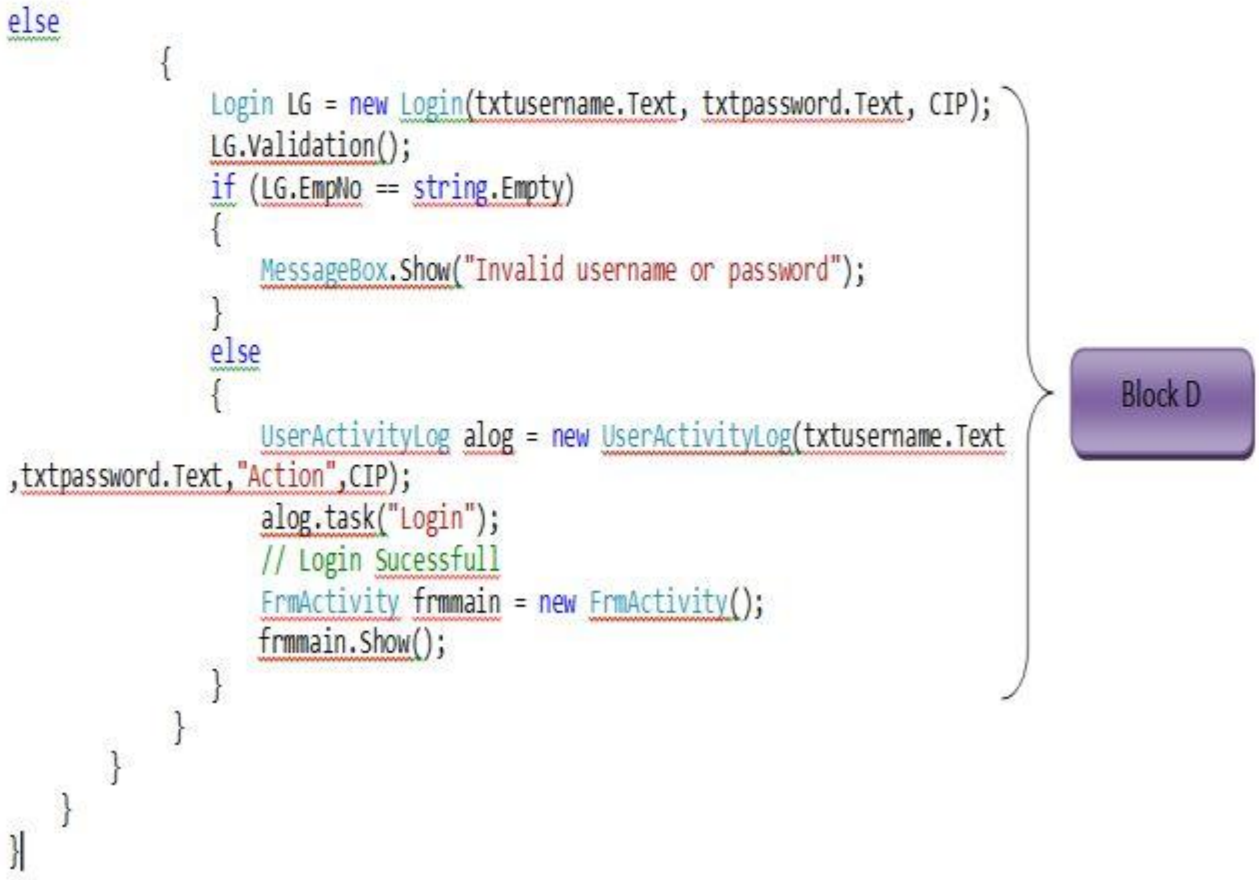
- A method task will be used to insert the data into the database, so we need to create the object on the main class with the nameof icon.
- Using an insert statement we can insert the new row into the database that will get the parameter action which can be "login", "add new", "save" etcetera.

Step 13

Please insert the following code in form1.



```
namespace UserActivityLog_OOP_Concept
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        protected  string CIP = "5";
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Location = new Point(258, 200);
            string host = Dns.GetHostName();
            IPHostEntry ip = Dns.GetHostEntry(host);
            CIP = (ip.AddressList[0].ToString());
            lblCP.Text = (ip.AddressList[0].ToString());
        }

        private void btnCancel_Click(object sender, EventArgs e)
        {
            Close();
        }

        private void BtnLogin_Click(object sender, EventArgs e)
        {
            if ((txtusername.Text == ""))
            {
                MessageBox.Show("Please Enter User Name");
            }
            else if (txtpassword.Text == "")
            {
                MessageBox.Show("Please Enter User Name");
            }
```



```
else
            {
                Login LG = new Login(txtusername.Text, txtpassword.Text, CIP);
                LG.Validation();
                if (LG.EmpNo == string.Empty)
                {
                    MessageBox.Show("Invalid username or password");
                }
                else
                {
                    UserActivityLog alog = new UserActivityLog(txtusername.Text
,txtpassword.Text,"Action",CIP);
                    alog.task("Login");
                    // Login Sucessfull
                    FrmActivity frmmain = new FrmActivity();
                    frmmain.Show();
                }
            }
        }
    }
}
```

Block A

- We have initialized the position of the form.
- String variable host Find host by name System.
- Gets a list of IP addresses that are associated with a host.
- Assigns the IP address to CIP.
- lblCP. Text is assigned the IP address to the label that will display the IP info.

Block B

- Close the login windows.

Block C

- Checking if textboxes are to not empty.

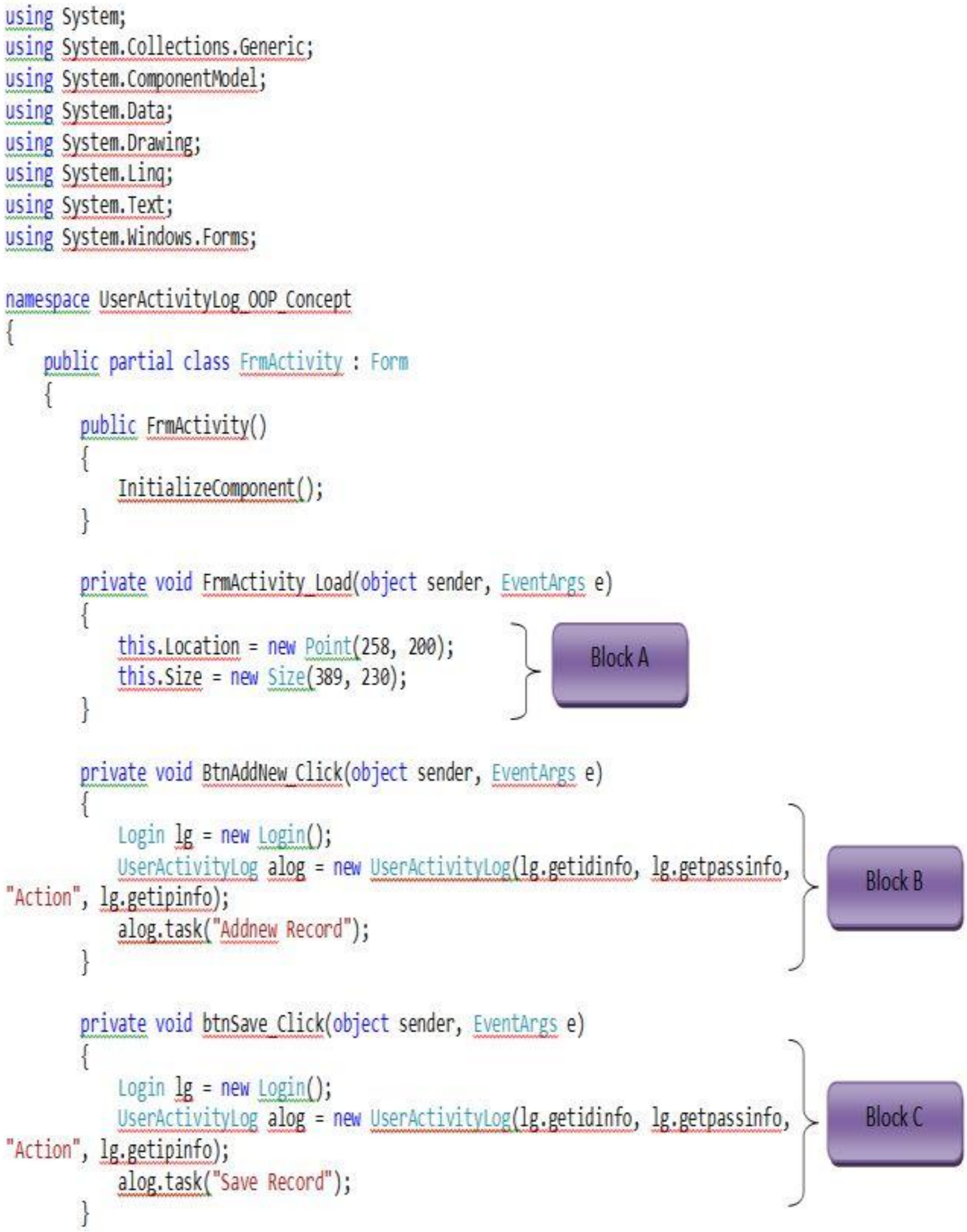
Block D

- Creating object LG that will activate the constructor of the class login with three parameters.
- Call the method validation using the object LG.
- Using "LG.EmpNo" public variable we check either user and password correct.
- Message box if information is not correct.

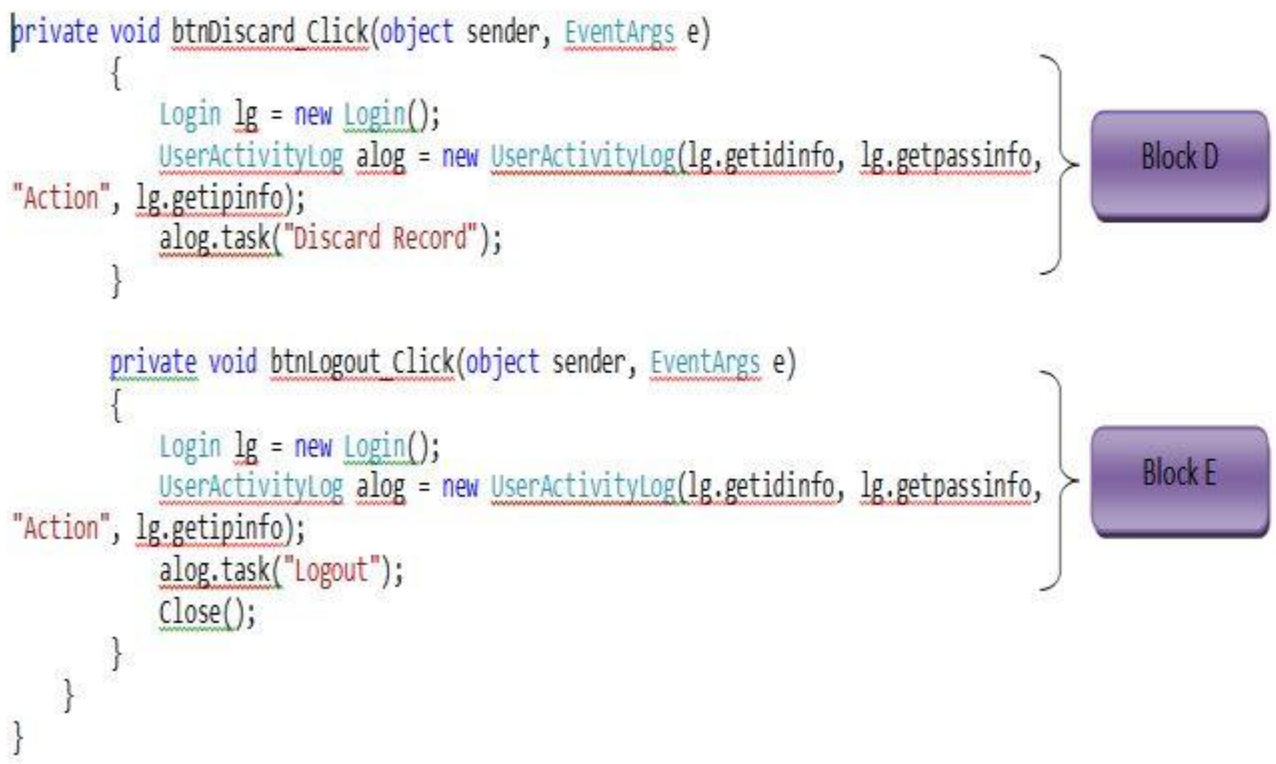
- If information is correct then we will create object "alog" of the class UserActivityLog; it calls the constructor with four parameters.
- Using the "alog" object we call the method task that will insert the data in the database.
- FrmActivity creates the object of the form.
- Load the form using the object name frmmain.

Step 14

Insert the following code into the FrmActivity form.



```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace UserActivityLog_OOP_Concept
{
    public partial class FrmActivity : Form
    {
        public FrmActivity()
        {
            InitializeComponent();
        }

        private void FrmActivity_Load(object sender, EventArgs e)
        {
            this.Location = new Point(258, 200);
            this.Size = new Size(389, 230);
        }

        private void BtnAddNew_Click(object sender, EventArgs e)
        {
            Login lg = new Login();
            UserActivityLog alog = new UserActivityLog(lg.getidinfo, lg.getpassinfo,
"Action", lg.getipinfo);
            alog.task("Addnew Record");
        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            Login lg = new Login();
            UserActivityLog alog = new UserActivityLog(lg.getidinfo, lg.getpassinfo,
"Action", lg.getipinfo);
            alog.task("Save Record");
        }
```



```
private void btnDiscard_Click(object sender, EventArgs e)
        {
            Login lg = new Login();
            UserActivityLog alog = new UserActivityLog(lg.getidinfo, lg.getpassinfo,
"Action", lg.getipinfo);
            alog.task("Discard Record");
        }

        private void btnLogout_Click(object sender, EventArgs e)
        {
            Login lg = new Login();
            UserActivityLog alog = new UserActivityLog(lg.getidinfo, lg.getpassinfo,
"Action", lg.getipinfo);
            alog.task("Logout");
            Close();
        }
    }
}
```

Block A

- We have initialized the position of the form .
- We create the object "alog" for the class UserActivityLog that will initialize member variables of the class UserActivityLog using three parameters and the values of the paramters are the properties of the login class.
- The next step is to use a method "task"of the class UserActivityLog with parameter "Addnew Record".

Block B

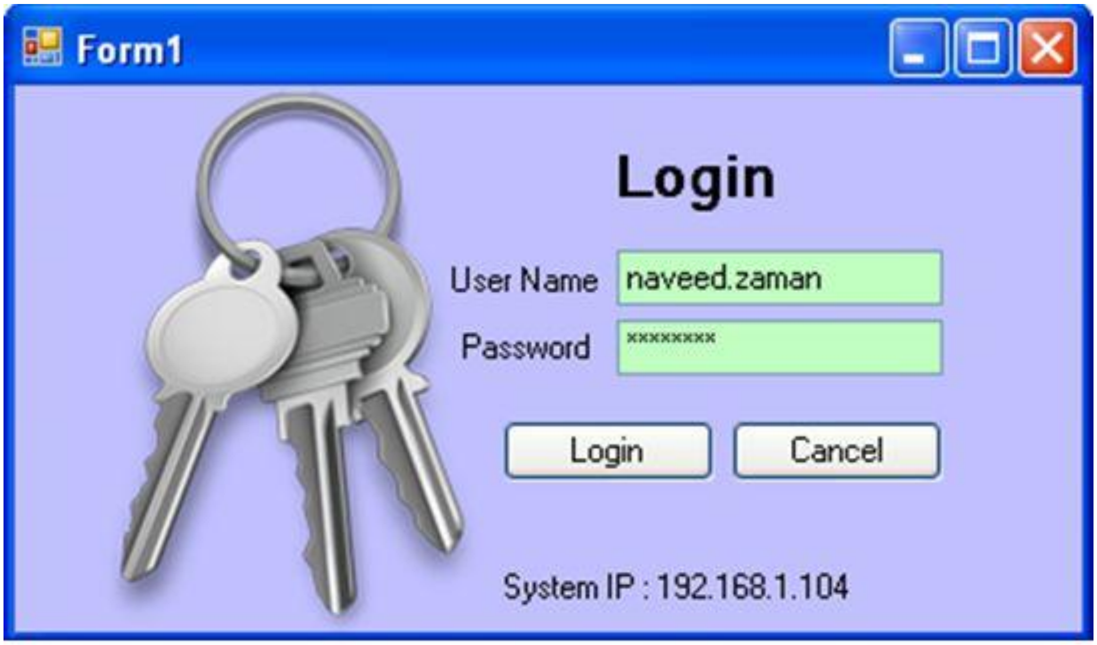
- We have initlized the position of the form .
- We create an object "alog" for the class UserActivityLog that will initialize the member variables of a class UserActivityLog using three parameters and the values of the paramter is the properties of the login class.
- The next step is to use the method "task"of the class UserActivityLog with the parameter "Save Record".

The same with the blocks, C, D and E.

Step 15

Now press F5. You will see the login screen; just enter:

user name: naveed.zaman
user name: khan@123

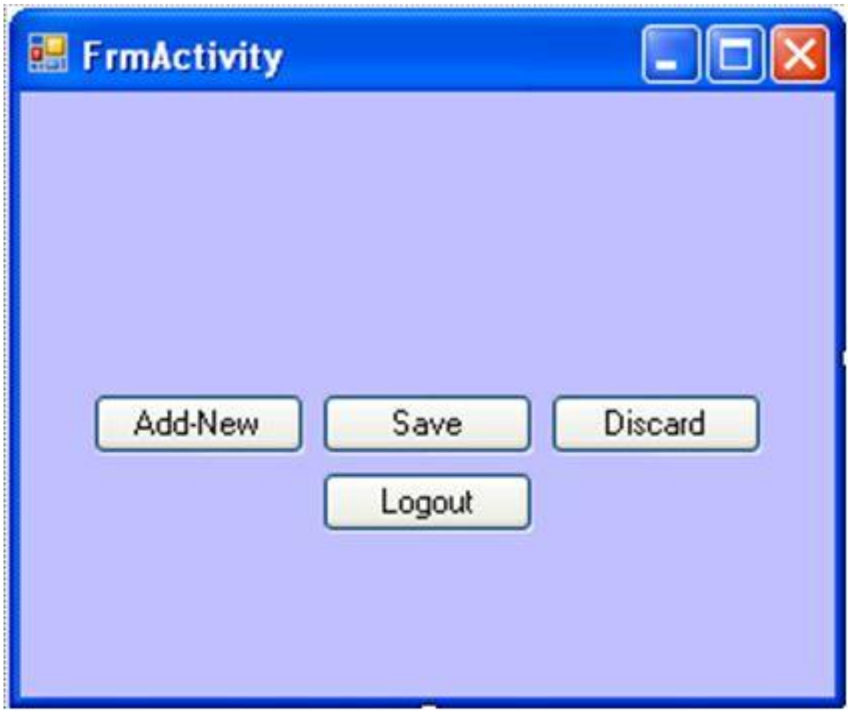


User Name naveed.zaman

Password *********

System IP : 192.168.1.104

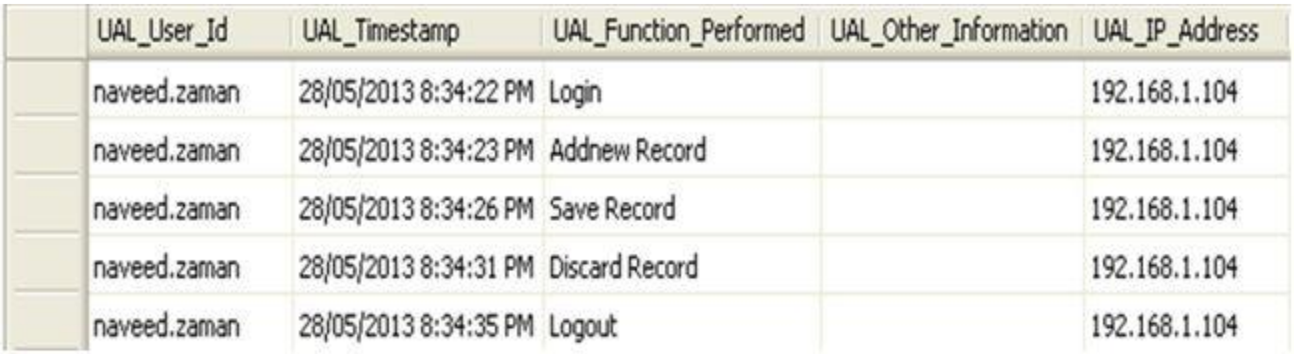
Click the login button.



Now press the "Add-New" button, "Save" button, "Discard" button and then press the "Login" button.

Step 16

Now open then SQL Server table; you will get a result like the following:



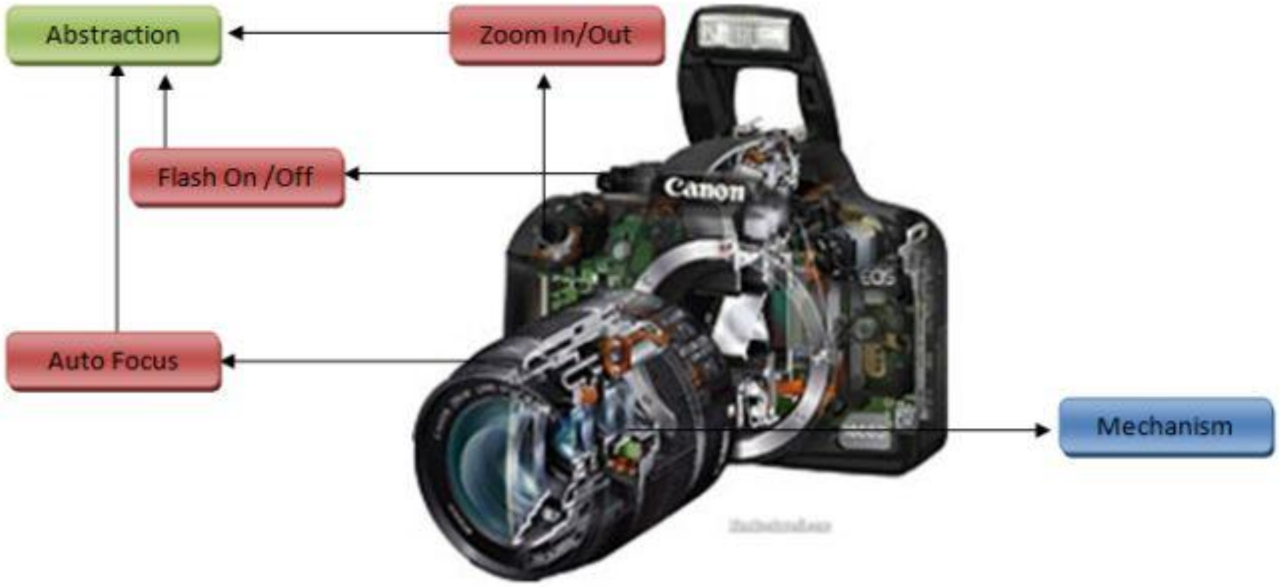
	UAL_User_Id	UAL_Timestamp	UAL_Function_Performed	UAL_Other_Information	UAL_IP_Address
	naveed.zaman	28/05/2013 8:34:22 PM	Login		192.168.1.104
	naveed.zaman	28/05/2013 8:34:23 PM	Addnew Record		192.168.1.104
	naveed.zaman	28/05/2013 8:34:26 PM	Save Record		192.168.1.104
	naveed.zaman	28/05/2013 8:34:31 PM	Discard Record		192.168.1.104
	naveed.zaman	28/05/2013 8:34:35 PM	Logout		192.168.1.104

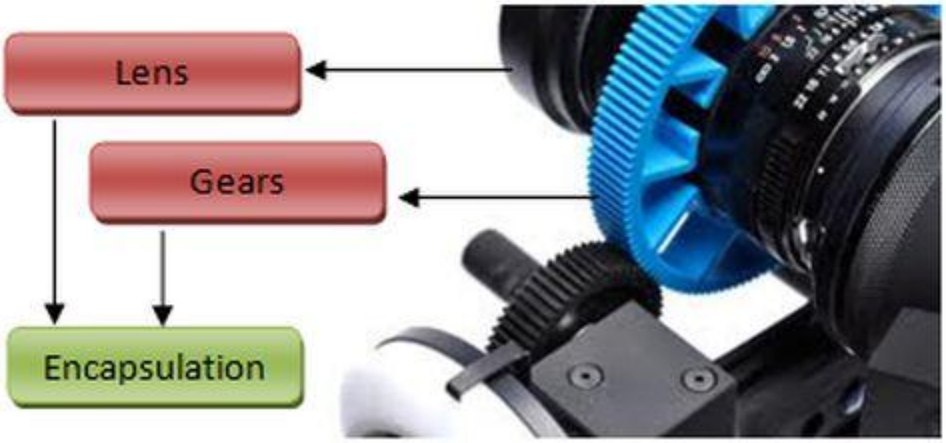
10. Difference between Encapsulation and Abstraction

There is a very basic difference between encapsulation and abstraction for beginners of OOP. They might get confused by it. But there is huge difference between them if you understand both the topics in detail.

Abstraction means to hide the unnecessary data from the user. The user only needs the required functionality or the output according to his requirements. For example a digital camera.

Dear reader, whenever we use a digital camera, we just click on the Zoom In and Zoom Out buttons and the camera zooms in and out but we can feel the lens moving. If we open the camera then we will see its complex mechanism that we can't understand. So pressing the button and getting the results according to your demand is the abstraction.





Encapsulation is simply combining the data members and functions into a single entity called an object.

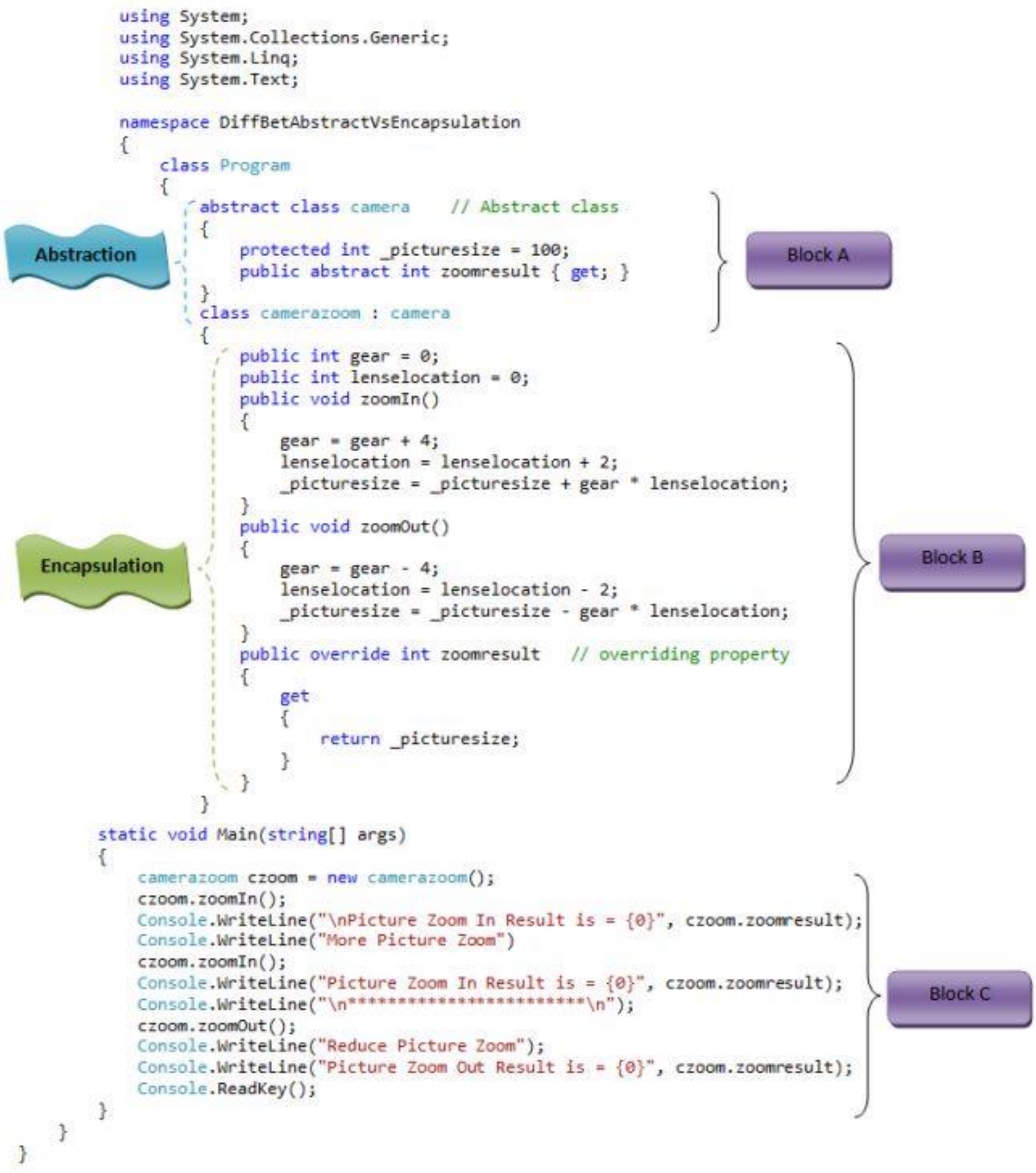
If we consider the camera example again, when we press zoom In/Out buttons, inside the camera it uses mechanisms that consists of gears and lenses to zoom in or zoom out. The combination of gears and lenses is called encapsulation that will help the zooming functionality to work smoothly.

In simple words we can say that "Abstraction is achieved through encapsulation".

Or

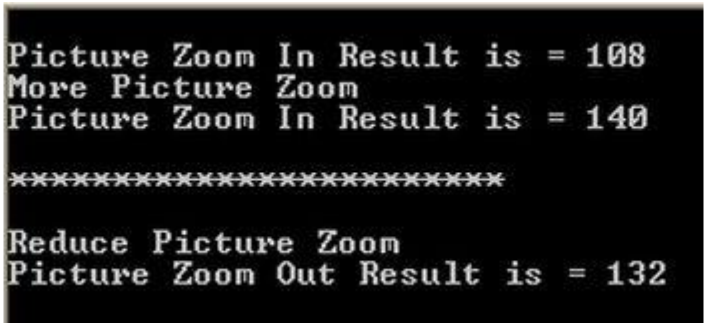
Abstraction solves the problem in the design side while encapsulation is the implementation.

Example:



```
          using System;
          using System.Collections.Generic;
          using System.Linq;
          using System.Text;

          namespace DiffBetAbstractVsEncapsulation
          {
              class Program
              {
                  abstract class camera     // Abstract class
                  {
                      protected int _picturesize = 100;
                      public abstract int zoomresult { get; }
                  }
                  class camerazoom : camera
                  {
                      public int gear = 0;
                      public int lenselocation = 0;
                      public void zoomIn()
                      {
                          gear = gear + 4;
                          lenselocation = lenselocation + 2;
                          _picturesize = _picturesize + gear * lenselocation;
                      }
                      public void zoomOut()
                      {
                          gear = gear - 4;
                          lenselocation = lenselocation - 2;
                          _picturesize = _picturesize - gear * lenselocation;
                      }
                      public override int zoomresult    // overriding property
                      {
                          get
                          {
                              return _picturesize;
                          }
                      }
                  }
        static void Main(string[] args)
        {
            camerazoom czoom = new camerazoom();
            czoom.zoomIn();
            Console.WriteLine("\nPicture Zoom In Result is = {0}", czoom.zoomresult);
            Console.WriteLine("More Picture Zoom")
            czoom.zoomIn();
            Console.WriteLine("Picture Zoom In Result is = {0}", czoom.zoomresult);
            Console.WriteLine("\n***********************\n");
            czoom.zoomOut();
            Console.WriteLine("Reduce Picture Zoom");
            Console.WriteLine("Picture Zoom Out Result is = {0}", czoom.zoomresult);
            Console.ReadKey();
        }
    }
}
```

Output:



```
Picture Zoom In Result is = 108
More Picture Zoom
Picture Zoom In Result is = 140

************************

Reduce Picture Zoom
Picture Zoom Out Result is = 132
```

Block A

- In this session we defined an abstract class.
- Having one protected variable and one function.

Block B

- In this session we have defined a class with an abstract class as the base class.
- With two public variables and two functions.
- One property zoomresult that is overriden from the base class
- In this block we have changed the value of the gear and lens in order to change the size of the _picturesize variable.

Block C

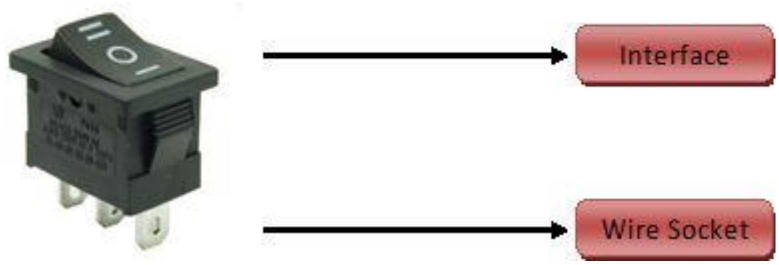
- In this session we have created the object of the class camerazoom.
- Then we have called the method zoom in
- After that we have shown the output of the variable _picturesize using property.
- Then we again call the method zoom in and show the results of the variable _picturesize .
- In the final part we have called the method zoom out the picture and checked the result.

11. Interface

Dear reader, today we will discuss another important component of OOP, interfaces. This topic again confuses OOP bginners. I have even found many questions in various forums regarding interfaces.

- What is an interface?
- When to use an interface?
- Why use an interface?
- Where to use an interface?

So I will explain interfaces with an example. When you press the "power" button of your machine, it's an interface between you and the electrical wiring on the other side of its plastic casing. You press the "power" button to turn the machine on and off.



In simple words, an interface defines how you can interact with a class, in other words what methods it supports.

What is an interface?

An interface is a programming structure/syntax that allows the computer to enforce certain properties on an object (class).

When to use an interface?

This is a very important question for the new developer. For example we have a core banking system. We all know that bank data is very sensitive. A little negligence is risky. So if the bank decided to develop a mobile banking web application from a third party developer then we can't provide full access to our main core banking application. So we must design the interface in our core banking application. You might use a DLL as well.

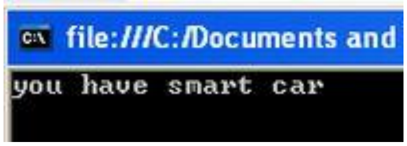
The other developers can use that DLL and send data for the transition in the main application. The third-party developers only access the core banking application with the limited rights that we have given them in the interface. So the interface is very useful in such conditions.

Simple Example

The following is the simple example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Interfacelesson
{
    class Program
    {
        interface ISupercar
        {
            /// Interface method declaration
            bool smartcar();
        }
         class mercedes :  ISupercar
        {
            public mercedes()
            {
            }
            /// Interface method definition in the class that implements it
            public bool smartcar()
            {
                Console.WriteLine("you have smart car");
                return true;
            }
        }
        static void Main(string[] args)
        {
            ISupercar mycar = new mercedes();
            mycar.smartcar();
            Console.Read();
        }
    }
}
```

Output:



```
you have smart car
```

Block A

Block A is:

- In this sessoion we have defined the interface ISupercar.
- Has one method, smartcar, without implementation.

Block B

Block B is:

- In this sessoion we have defined the class Mercedes with the implemented interface ISupercar.
- Having one method smartcar with implementation.

Block C

Block C is:

- Create one object of the interface using "ISupercar mycar = new mercedes();".
- Using that object "mycar" we access the method that we have defined in the class mercedes.

Why use an interface?

- Create loosely coupled software
- Support design by contract (an implementer must provide the entire interface)

- Allow for pluggable software
- Allow objects to interact easily
- Hide implementation details of classes from each other
- Facilitate reuse of software

Where to use an interface?

C# does not support multiple inheritances. An interface on the other hand resolves the multiple inheritances problem. One class can implement **many** interfaces.

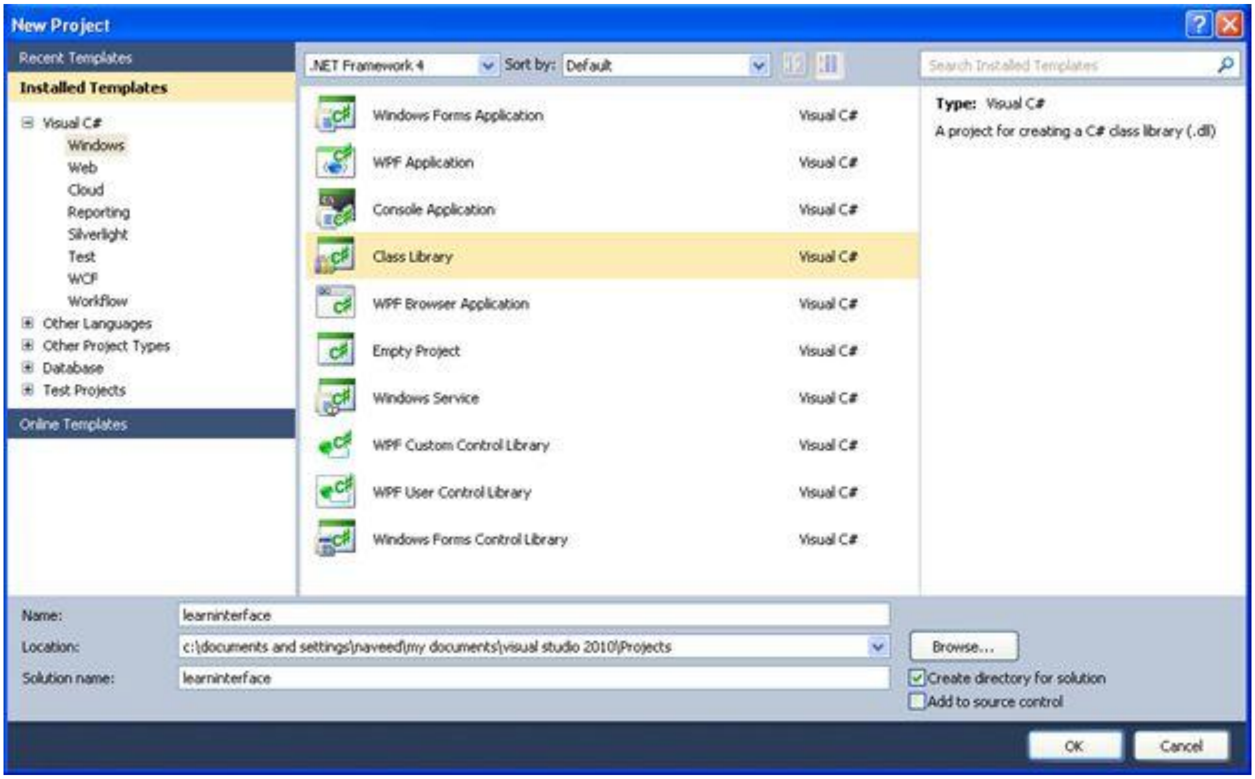
An Advance Example

In this example we will create the two small projects. In the first project we will create a DLL file and define one interface. In the second project we will call the DLL file that we have designed in the first project, then we will use the interface of the first project. This will help us to understand the concept of interfces more clearly.

In this project our concept is very simple. We will create the core banking application concept and then we will create the DLL of the core banking project and use it in the mobilebanking project.

Step 1:

Create the new project with the name of "learninterface" with project type Class Library.



New Project
Recent Templates
Installed Templates
Visual C#
Windows
Web
Cloud
Reporting
Silverlight
Test
WCF
Workflow
Other Languages
Other Project Types
Database
Test Projects
Online Templates
Windows Forms Application Visual C#
WPF Application Visual C#
Console Application Visual C#
Class Library Visual C#
WPF Browser Application Visual C#
Empty Project Visual C#
Windows Service Visual C#
WPF Custom Control Library Visual C#
WPF User Control Library Visual C#
Windows Forms Control Library Visual C#
Search Installed Templates
Browse...
OK
Cancel

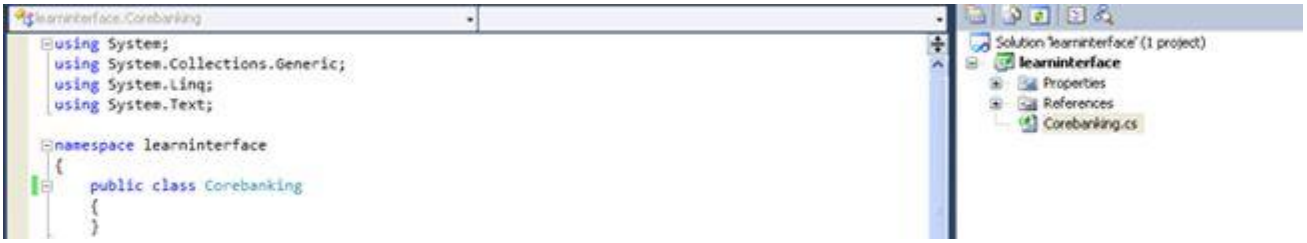
.NET Framework 4 Sort by: Default

Type: Visual C#
A project for creating a C# class library (.dll)

Name: learninterface
Location: c:\documents and settings\naveed\my documents\visual studio 2010\Projects
Solution name: learninterface
☑ Create directory for solution
□ Add to source control

Step 2:

We will rename Class1 to Corebanking.



learninterface.Corebanking

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace learninterface
{
    public class Corebanking
    {
    }
```

Solution 'learninterface' (1 project)
learninterface
Properties
References
Corebanking.cs

Step 3:

Now we will define the interface with the name of "Icorebanking" with the one method "void updatetransition(int a, int b);" without implementation because an interface does not allow implementation of the method in the interface and it's a good practice to always use the capital "I" for the interface name.

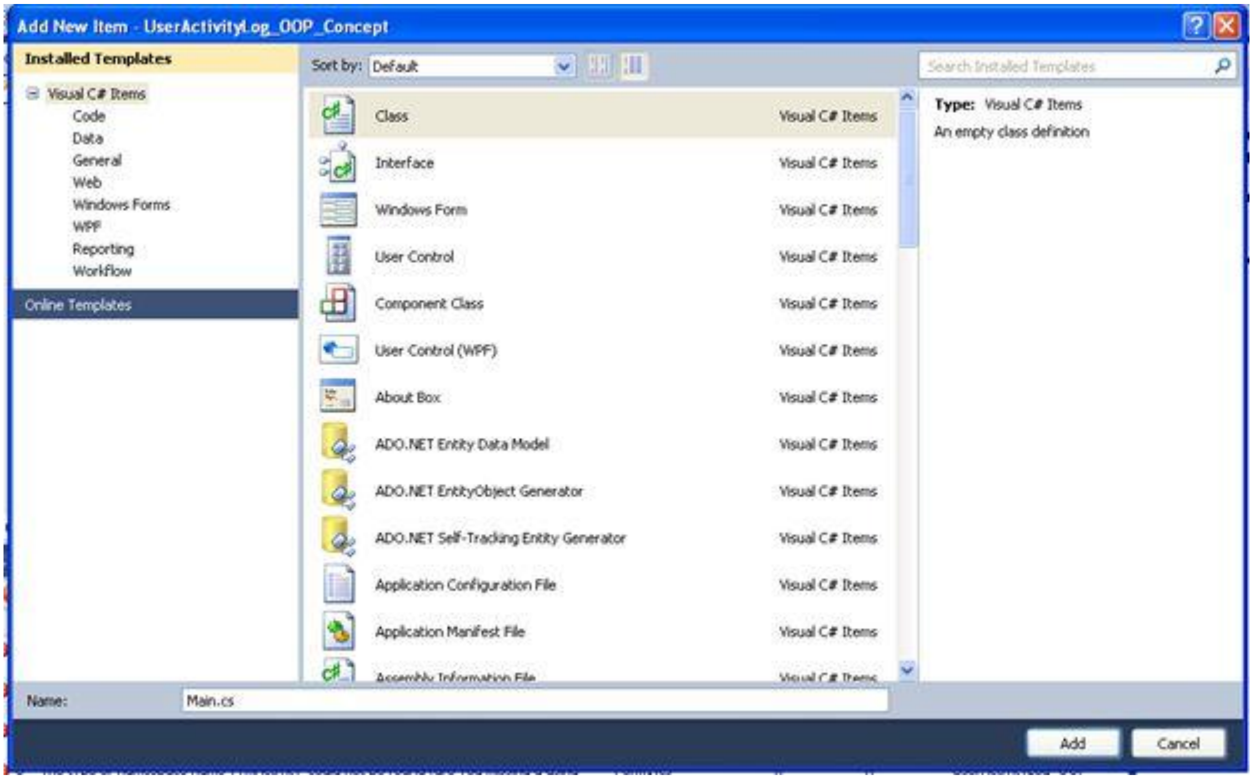
Step 4:

Now we will implement the interface in a class Corebanking. So we must implement all the methods of the interface in the class Corebanking.

```
public interface Icorebanking
{
   void updatetransition(string cusId, string Ref, int amtdeposit, int amtcredit);
}
public class Corebanking : Icorebanking
{
   public void updatetransition(string cusId, string Ref, int amtdeposit, int amtcredit)
   {
       Transition trans = new Transition();
       trans.save(cusId, Ref, amtdeposit, amtcredit);
  }
}
```

Step 5:

Add the new class with the name "Main".


Add New Item - UserActivityLog_OOP_Concept
Installed Templates
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
Reporting
Workflow
Online Templates
Sort by: Default
Search Installed Templates
Class
Interface
Windows Form
User Control
Component Class
User Control (WPF)
About Box
ADO.NET Entity Data Model
ADO.NET EntityObject Generator
ADO.NET Self-Tracking Entity Generator
Application Configuration File
Application Manifest File
Assembly Information File
Visual C# Items
Type: Visual C# Items
An empty class definition
Add
Cancel

Name: Main.cs

Step 6:

Now we will create the function GetDBConnection as a public function so it can be accessed from anywhere in the project. But you must modify the data source setting according to your own settings.

For Example:

1. SQL Server instance name. (.\\CRMIS)
2. SQL Server user name and password. (user id sa password.###Reno123)
3. Initial Catalog=PSH (database name)

Add the public static class in the Main.cs:

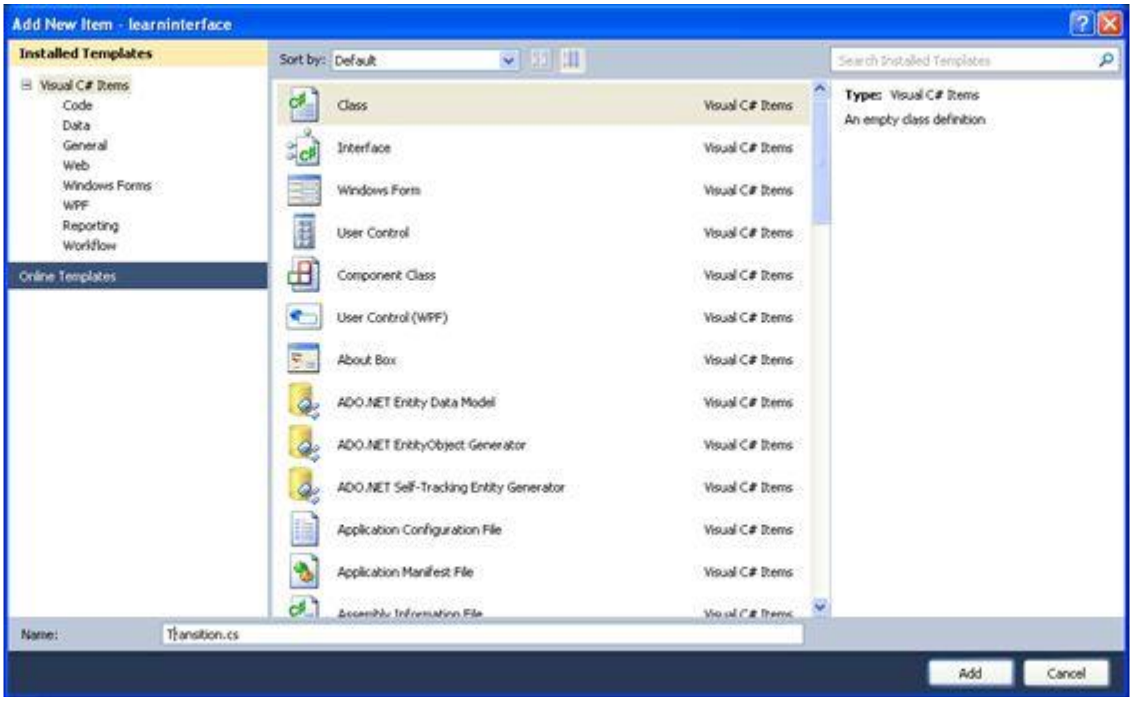
```
public static SqlConnection GetDBConnection()
{
    SqlConnection conn = new SqlConnection(
    "Data Source=.\\CRMIS;Initial Catalog=PSH;User ID=sa;Password=###Reno321");
    return conn;
}
```

Step 7:

```
CREATE TABLE [dbo].[Transition](
[TRA_Customer_Id] [varchar](50) NULL,
[TRA_Account_Ref] [varchar](15) NULL,
[TRA_Amount_Deposit] [money] NULL,
[TRA_Amount_Credit] [money] NULL
) ON [PRIMARY]
```

Step 8:

Add the new class with the name "Transition" and insert the following method into it.


Add New Item - learninterface
Installed Templates
Visual C# Items
Code
Data
General
Web
Windows Forms
WPF
Reporting
Workflow
Online Templates
Sort by: Default
Search Installed Templates
Class
Interface
Windows Form
User Control
Component Class
User Control (WPF)
About Box
ADO.NET Entity Data Model
ADO.NET EntityObject Generator
ADO.NET Self-Tracking Entity Generator
Application Configuration File
Application Manifest File
Assembly Information File
Visual C# Items
Type: Visual C# Items
An empty class definition
Add
Cancel

Name: Transition.cs

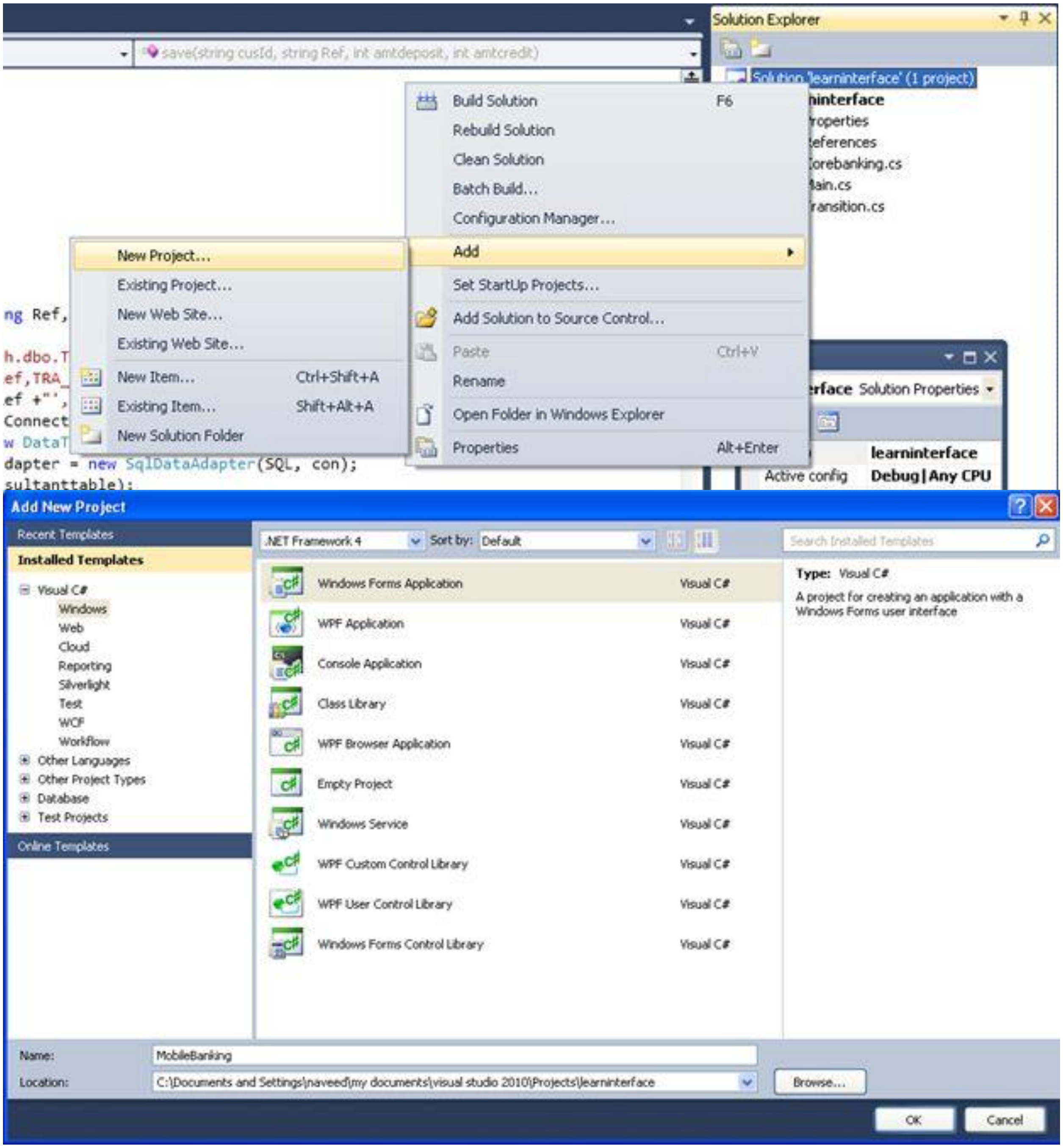
Step 9:

Now add the method into the Transition class.

```
public void save(string cusId,string Ref,int amtdeposit,int amtcredit)
{
    string SQL = @" insert into psh.dbo.Transition
    (TRA_Customer_Id,TRA_Account_Ref,TRA_Amount_Deposit,TRA_Amount_Credit)
    values( '" + cusId + "'      ,'"+ Ref +"','" + amtdeposit + "','" + amtcredit + "')";
    SqlConnection con = Main.GetDBConnection();
    DataTable consultanttable = new DataTable();
    SqlDataAdapter Consultantdataadapter = new SqlDataAdapter(SQL, con);
    Consultantdataadapter.Fill(consultanttable);
}
```

Step 10:

Now add another project as shown in the diagram with the name of MobileBanking:



```
save(string custId, string Ref, int amtdeposit, int amtcredit)
```

```
ng Ref,
h.dbo.T
ef,TRA_
ef +"',
Connect
w DataT
dapter = new SqlDataAdapter(SQL, con);
sultanttable);
```

learninterface
Active config Debug|Any CPU

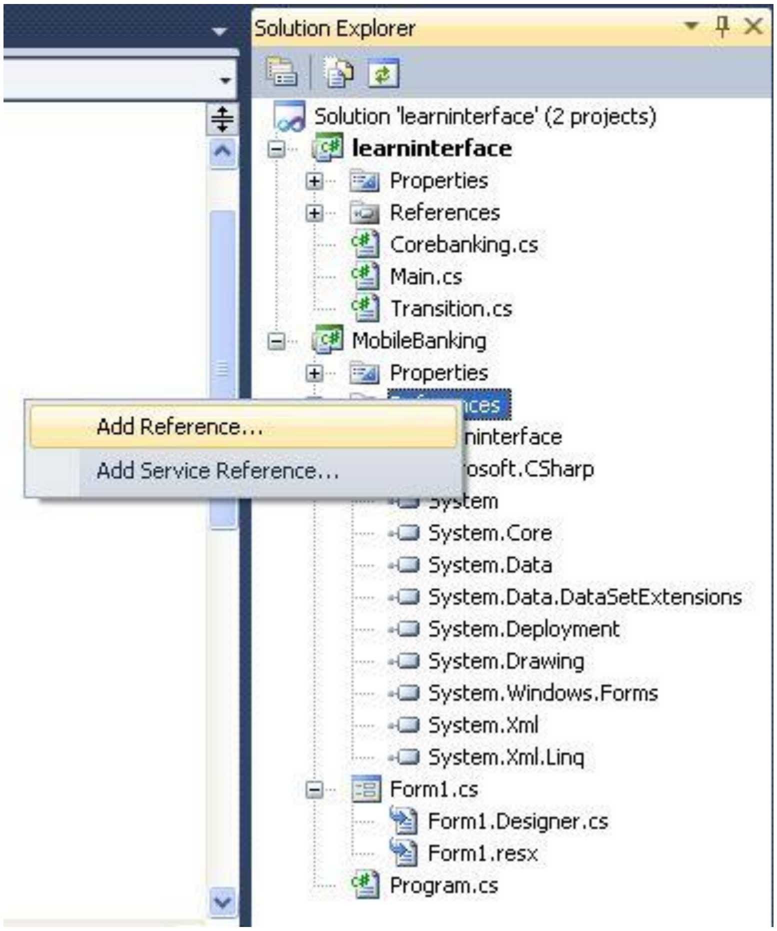
Type: Visual C#

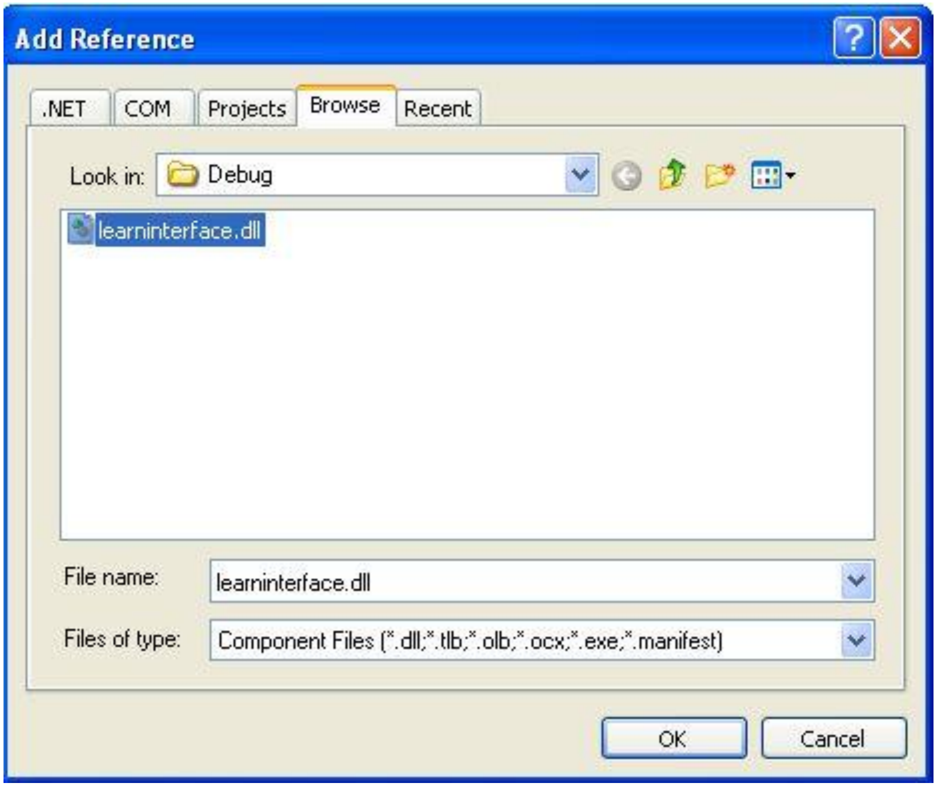
A project for creating an application with a Windows Forms user interface

Name: MobileBanking

Location: C:\Documents and Settings\naveed\my documents\visual studio 2010\Projects\learninterface

Step 11:





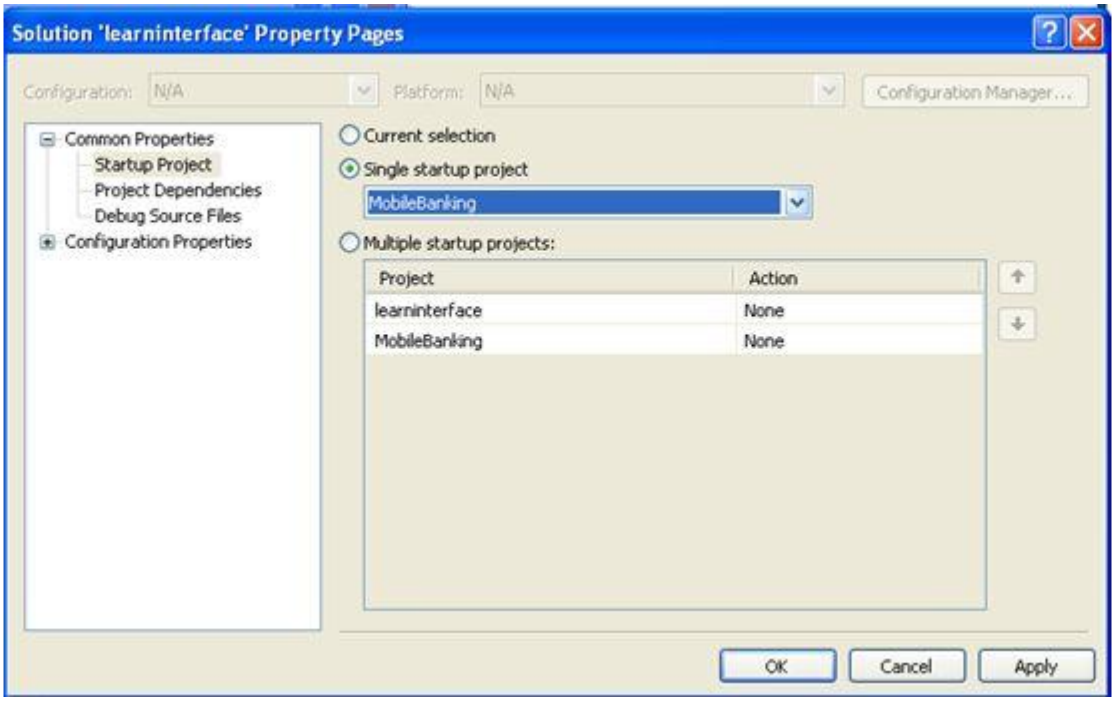
Look in: Debug

File name: learninterface.dll

Files of type: Component Files (*.dll;*.tlb;*.olb;*.ocx;*.exe;*.manifest)

It's an important step. First we must add the referance of the DLL file to the MobileBanking project, then we will design the form1.

Step 12:



Configuration: N/A

Platform: N/A

○ Current selection

● Single startup project

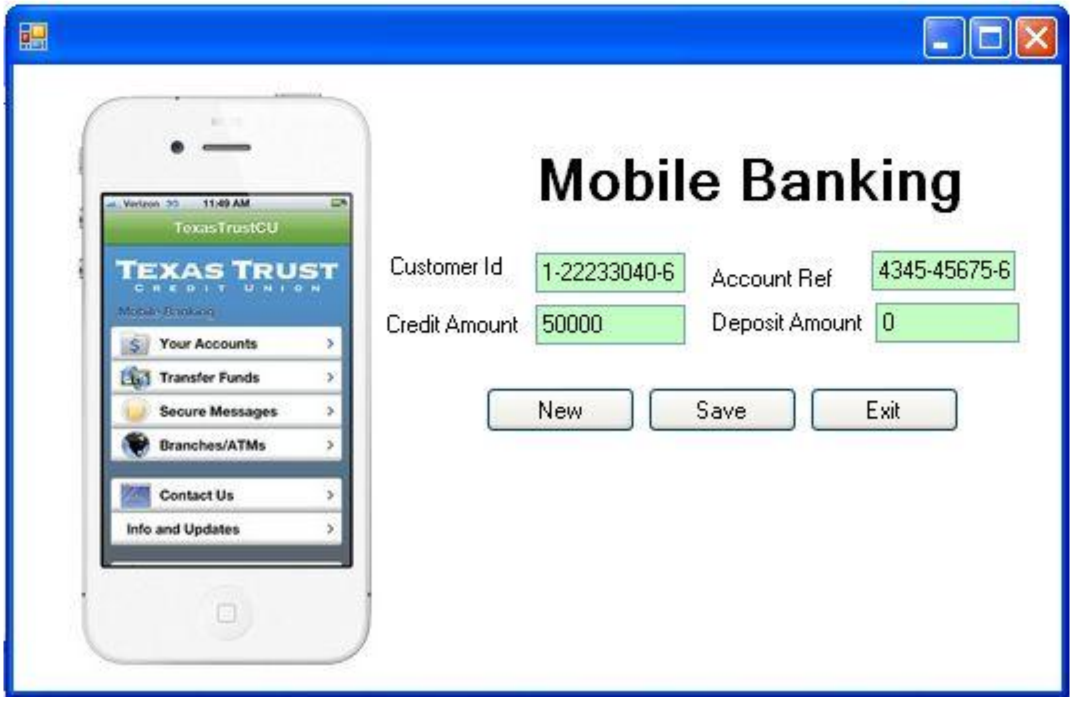
MobileBanking

○ Multiple startup projects:

Project	Action
learninterface	None
MobileBanking	None

Set the project startup type.

Step 13:

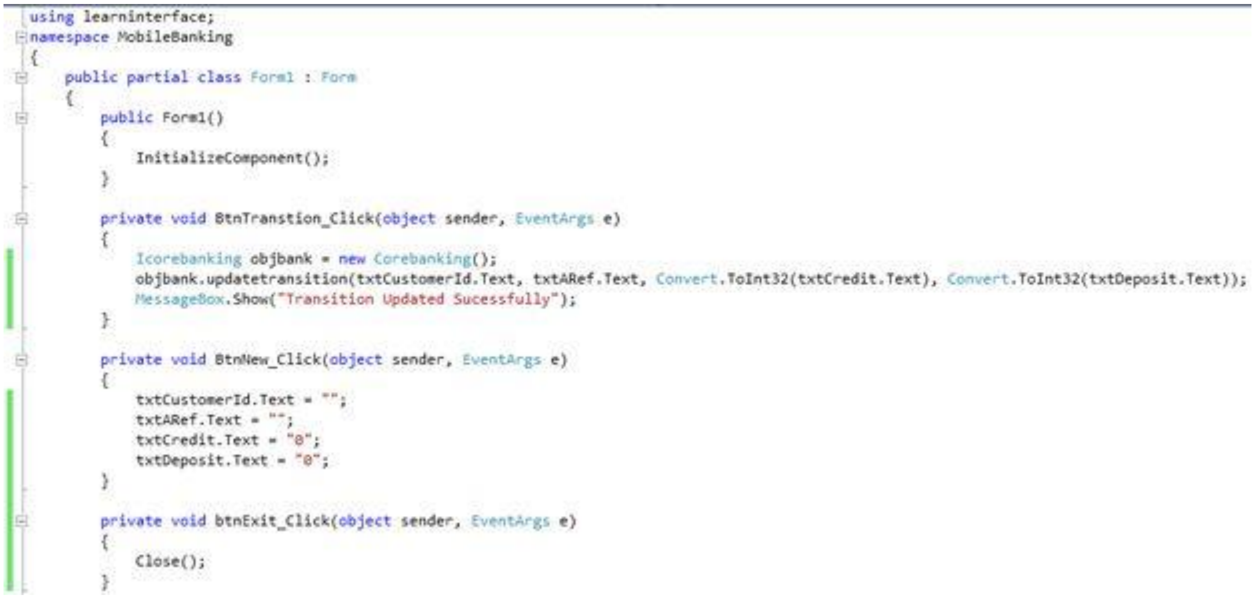


Mobile Banking

Customer Id	1-22233040-6	Account Ref	4345-45675-6
Credit Amount	50000	Deposit Amount	0

Now we need to design the form as shown in the picture to send data into the datecase using the interface DLL created in the first project.

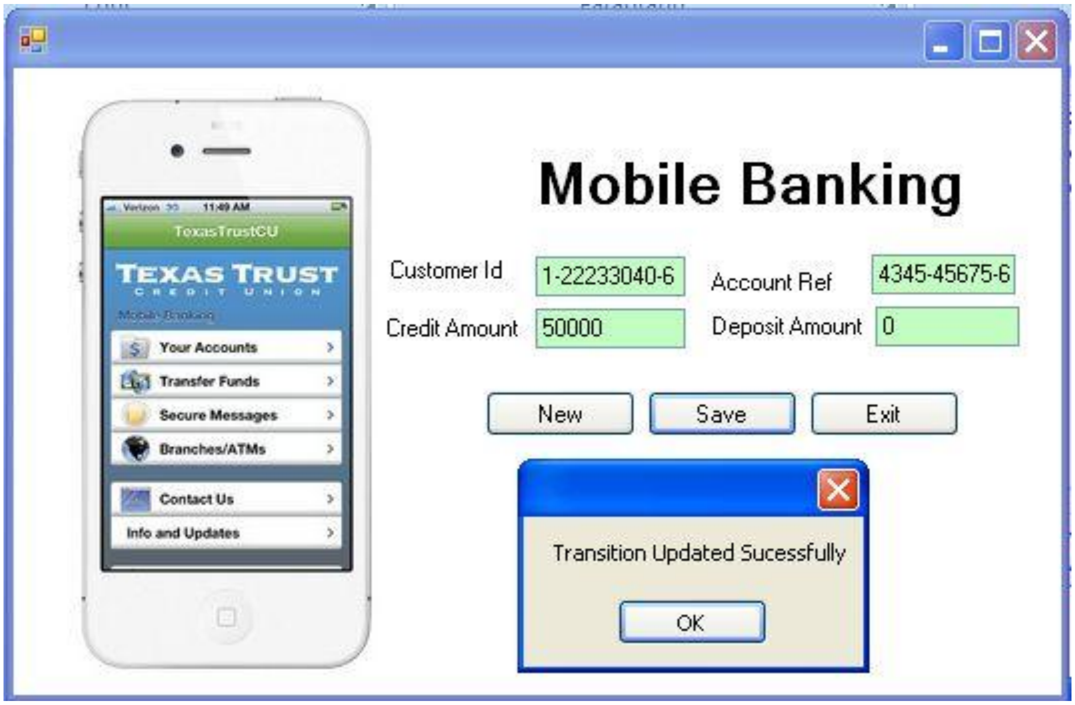
Step 14:



```
using learninterface;
namespace MobileBanking
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void BtnTranstion_Click(object sender, EventArgs e)
        {
            Icorebanking objbank = new Corebanking();
            objbank.updatetransition(txtCustomerId.Text, txtARef.Text, Convert.ToInt32(txtCredit.Text), Convert.ToInt32(txtDeposit.Text));
            MessageBox.Show("Transition Updated Sucessfully");
        }

        private void BtnNew_Click(object sender, EventArgs e)
        {
            txtCustomerId.Text = "";
            txtARef.Text = "";
            txtCredit.Text = "0";
            txtDeposit.Text = "0";
        }

        private void btnExit_Click(object sender, EventArgs e)
        {
            Close();
        }
```

In that step we will insert the following code into the button.

Step 15:



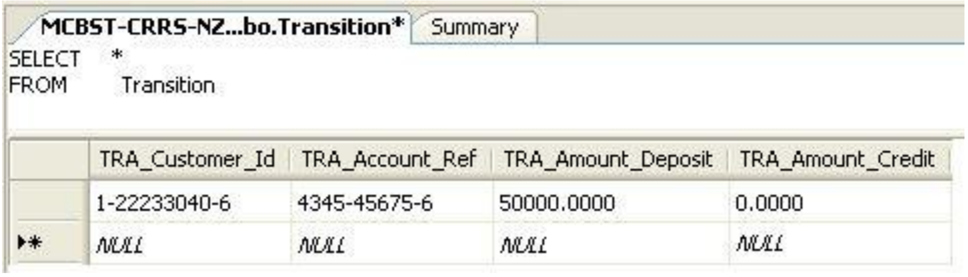
Mobile Banking

Customer Id 1-22233040-6
Account Ref 4345-45675-6
Credit Amount 50000
Deposit Amount 0

Transition Updated Sucessfully

Now execute the project and enter the following data into the textboxes and click the Save button.

Step 16:



```
SELECT    *
FROM      Transition
```

	TRA_Customer_Id	TRA_Account_Ref	TRA_Amount_Deposit	TRA_Amount_Credit
	1-22233040-6	4345-45675-6	50000.0000	0.0000
▸*	NULL	NULL	NULL	NULL

Now check the database and you will see the following result.

12. Virtual Methods

Dear reader today we will discuss another important component of OOP, virtual methods. I will explain this using a simple example. Fig (1.0) shows Class A having one virtual method AA and its implementation is a shirt with a yellow flower. Class B inherits Class A and overrides the method AA; its implementation is different since it wears a different shirt style with cap.

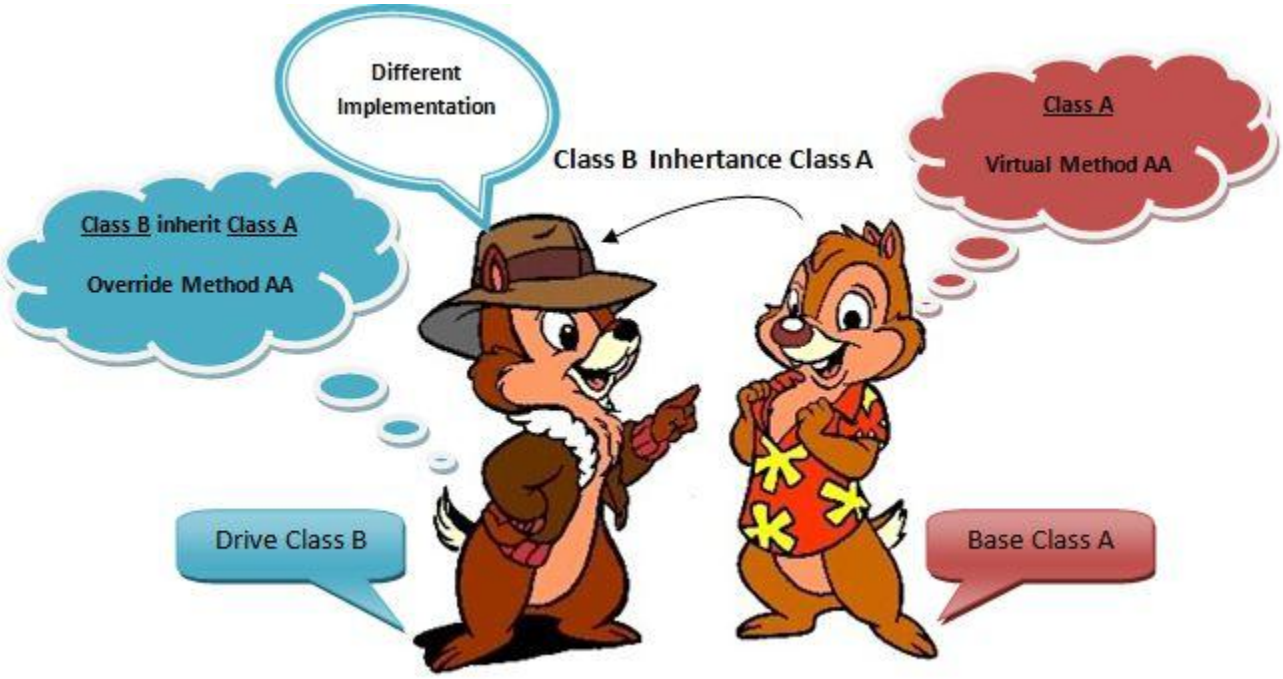
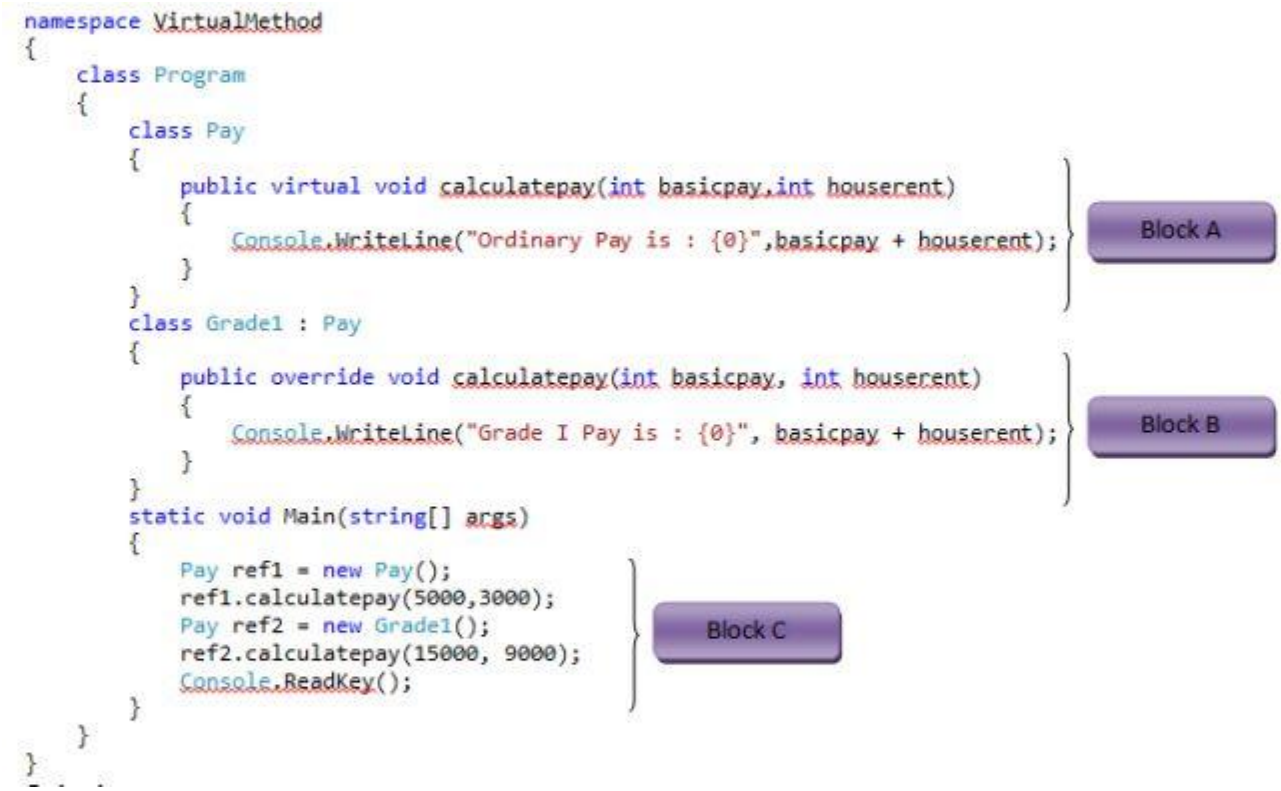


Fig (1.0)(Simple Virtual Example)

In simple words:

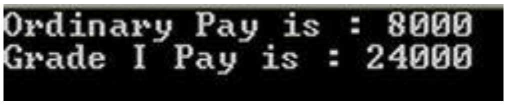
A virtual method is a method that can be redefined in derived classes. A virtual method has an implementation in a base class as well as in a derived class. It is used when a method's basic functionality is the same but sometimes more functionality is needed in the derived class.

Simple Example



```
namespace VirtualMethod
{
    class Program
    {
        class Pay
        {
            public virtual void calculatepay(int basicpay,int houserent)
            {
                Console.WriteLine("Ordinary Pay is : {0}",basicpay + houserent);
            }
        }
        class Grade1 : Pay
        {
            public override void calculatepay(int basicpay, int houserent)
            {
                Console.WriteLine("Grade I Pay is : {0}", basicpay + houserent);
            }
        }
        static void Main(string[] args)
        {
            Pay ref1 = new Pay();
            ref1.calculatepay(5000,3000);
            Pay ref2 = new Grade1();
            ref2.calculatepay(15000, 9000);
            Console.ReadKey();
        }
    }
}
```

Output:



```
Ordinary Pay is : 8000
Grade I Pay is : 24000
```

Block A:

- In this sessoion we have defines the class Pay.
- In that Pay class we have one virtual method calculatepay.
- Now we have the implementation of that calculatepay method that will calculate and print the pay.

Block B:

- In this session we have defined the class Grade1 that inherts the class pay.
- After that we have overriden the class calculatepay having the same signature.
- Now we have the implementation of that calculatepay method that will calculate and print the pay.

Block C:

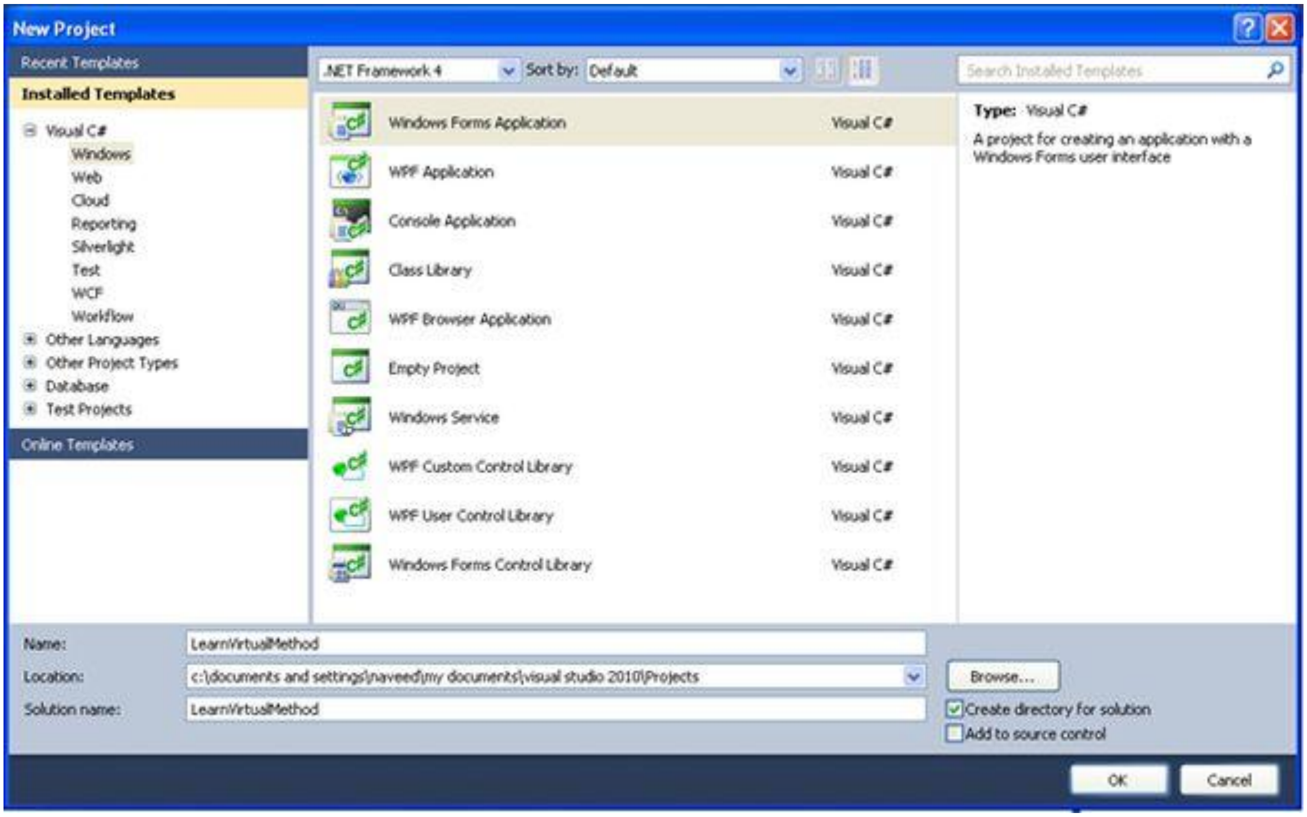
- First we have defined the object of the class pay, then we call the methods of the class pay with two arguments that will calculate the ordinary pay.
- Then we will again create the object of the Grade1 class and using that object we call overrode the method calculatepay.

Real Example:

In this example we will create the small application that will calculate the tax of the gross pay. In our application we allow the tax exemption for employees whose age is greater than 50 years.

Step 1:

Create the new project with the name of LearnVirtualMethod.



.NET Framework 4 Sort by: Default

Type: Visual C#

A project for creating an application with a Windows Forms user interface

Name: LearnVirtualMethod

Location: c:\documents and settings\naveed\my documents\visual studio 2010\Projects

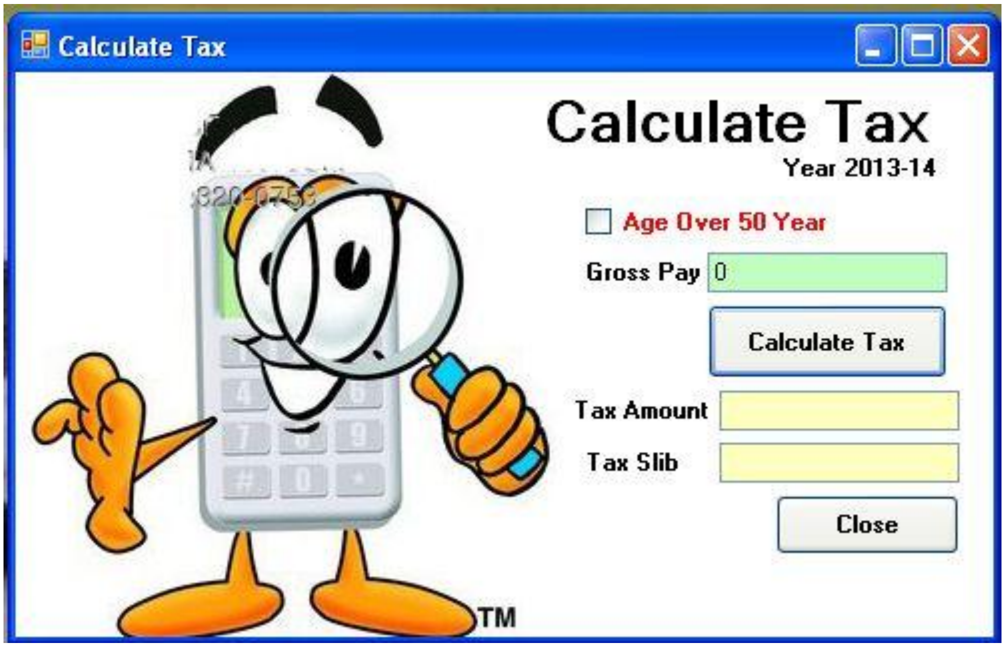
Solution name: LearnVirtualMethod

☑ Create directory for solution

□ Add to source control

Step 2:

Design the form1 as shown in the picture.



Calculate Tax

Year 2013-14

□ Age Over 50 Year

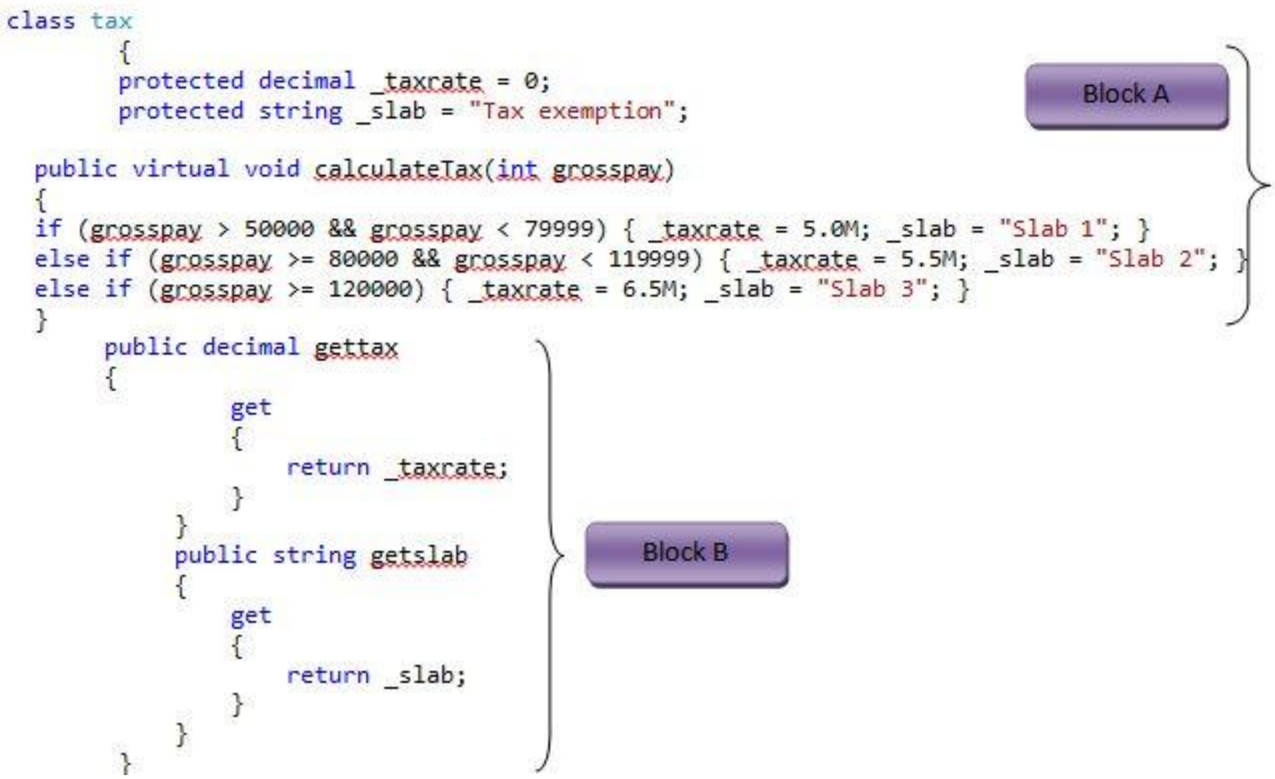
Gross Pay 0

Tax Amount

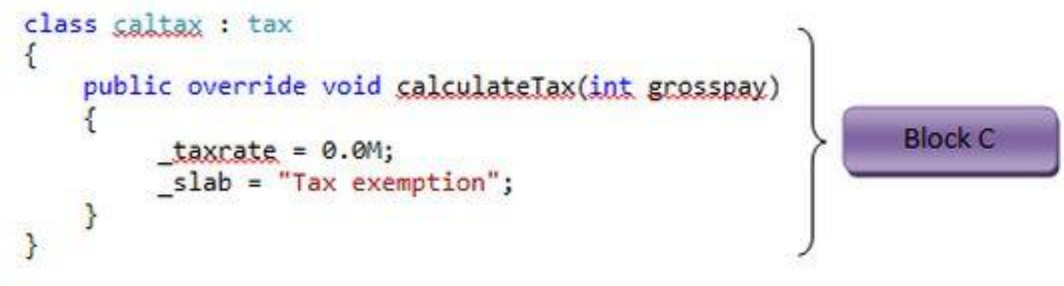
Tax Slib

Step 3:

Create a new class with name of tax as in the following. Having one virtual method calculateTax.



```
class tax
        {
        protected decimal _taxrate = 0;
        protected string _slab = "Tax exemption";

  public virtual void calculateTax(int grosspay)
  {
  if (grosspay > 50000 && grosspay < 79999) { _taxrate = 5.0M; _slab = "Slab 1"; }
  else if (grosspay >= 80000 && grosspay < 119999) { _taxrate = 5.5M; _slab = "Slab 2"; }
  else if (grosspay >= 120000) { _taxrate = 6.5M; _slab = "Slab 3"; }
  }
       public decimal gettax
       {
                get
                {
                    return _taxrate;
                }
            }
            public string getslab
            {
                get
                {
                    return _slab;
                }
            }
        }
```



```
class caltax : tax
{
    public override void calculateTax(int grosspay)
    {
         _taxrate = 0.0M;
         _slab = "Tax exemption";
    }
}
```

Block A:

- In this session we have defined the class tax.
- In that tax class we have two protected variables.
- Now we have created one virtual method, calculateTax.
- Then we have implemented the method calculateTax using an if statement that will assign the values to the variables.

Block B:

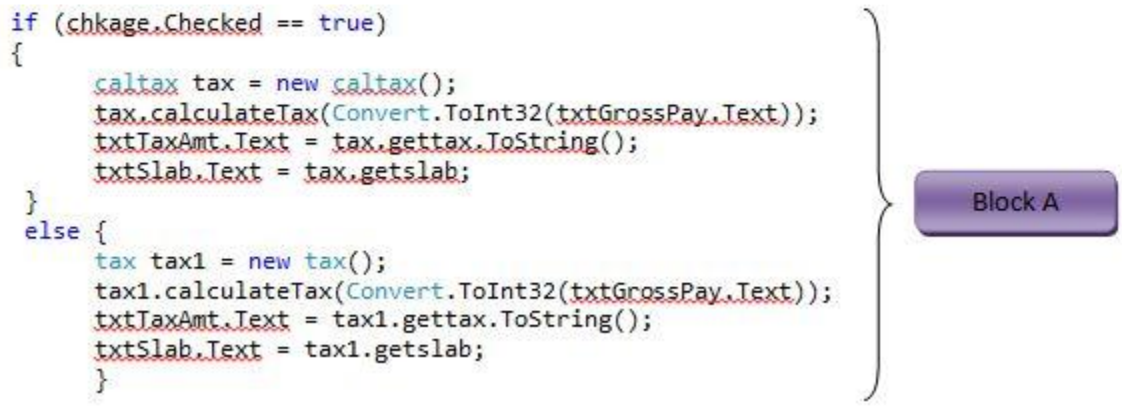
- In this session we have defined the two gettax and getslab properties that will help to get the result.

Block C:

- In this session we have defined the class caltax that inherts the class tax.
- After that we have overriden the class calculateTax having the same signature.
- Now we need to implement the calculateTax method to calculate the tax for the employee whose ages is greater the 50 years.

Step 4:

Enter the following code in the (Calculate Tax) button.



```
if (chkage.Checked == true)
{
      caltax tax = new caltax();
      tax.calculateTax(Convert.ToInt32(txtGrossPay.Text));
      txtTaxAmt.Text = tax.gettax.ToString();
      txtSlab.Text = tax.getslab;
 }
 else {
      tax tax1 = new tax();
      tax1.calculateTax(Convert.ToInt32(txtGrossPay.Text));
      txtTaxAmt.Text = tax1.gettax.ToString();
      txtSlab.Text = tax1.getslab;
      }
```

Block A:

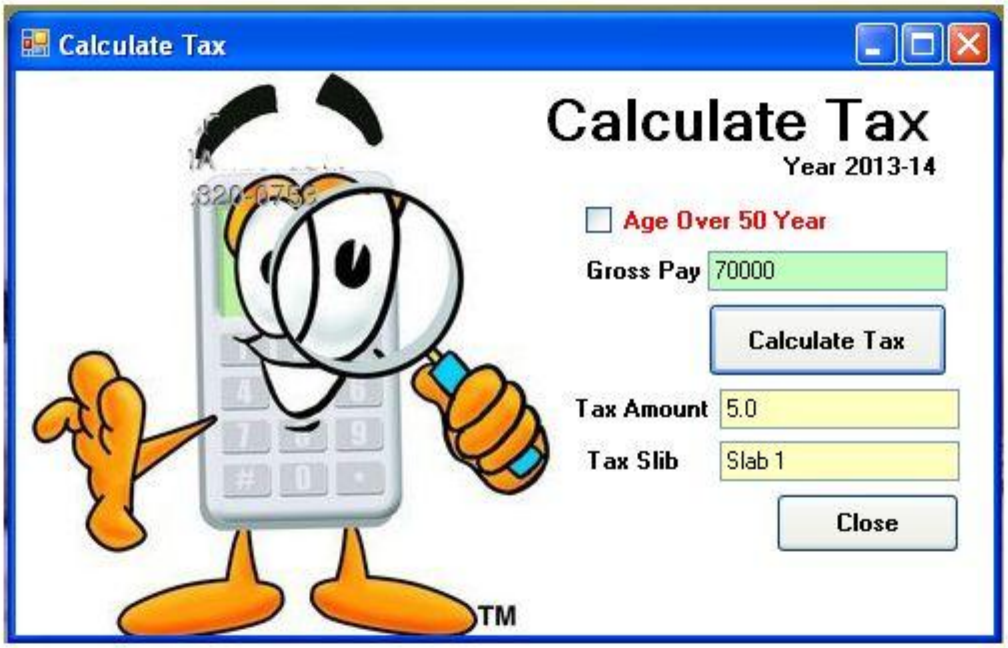
- In this session we have used an if statement that will determeine whether to create the object of the virtual method or overriden method.

- Then it will calculate the tax and return the result in text boxes.

Step 5:

To execute the program press F5 and enter the gross pay 70000 and click on the button.

The calculated tax will be shown as in the following result.



□ Age Over 50 Year

Gross Pay: 70000

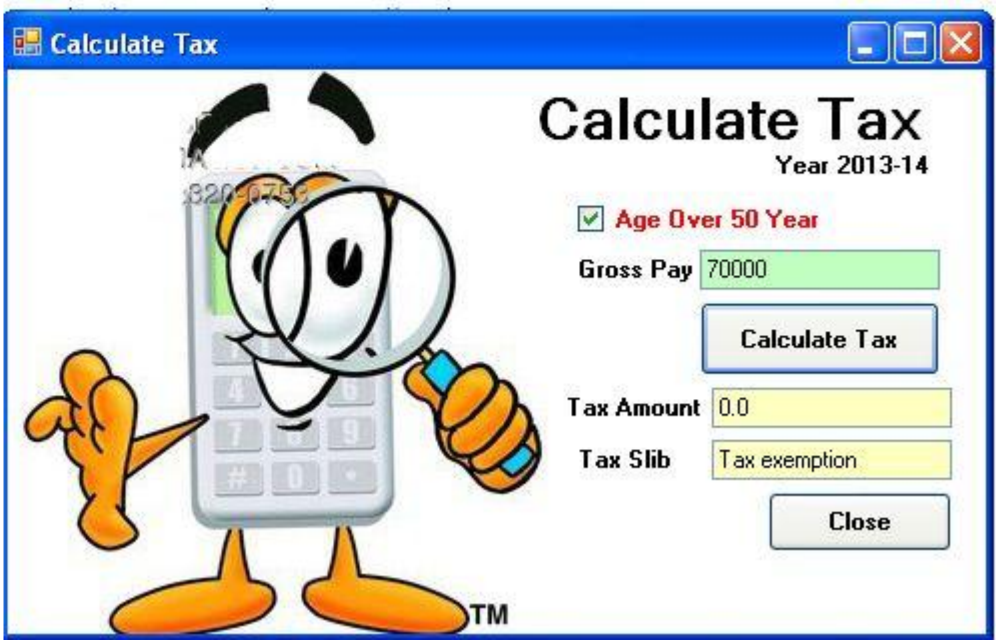
Tax Amount: 5.0

Tax Slib: Slab 1

Step 6:

Now click the check box button and again click the click button Calculate tax.

You will get the following result.



Calculate Tax

Calculate Tax
Year 2013-14

☑ Age Over 50 Year

Gross Pay: 70000

Tax Amount: 0.0

Tax Slib: Tax exemption

13. Delegates

Dear reader, today we will discuss another important component of OOP, which is delegates. It's the modern approach to call methods. Using that approach you can call any method with the same signature as the delegate. So we can say that a delegate is the launching pad for methods; any method can be launched that matches the pattern. Or a Delegate is a class. When you create an instance of it, you pass in the function name (as a parameter for the delegate's constructor) to which this delegate will refer.

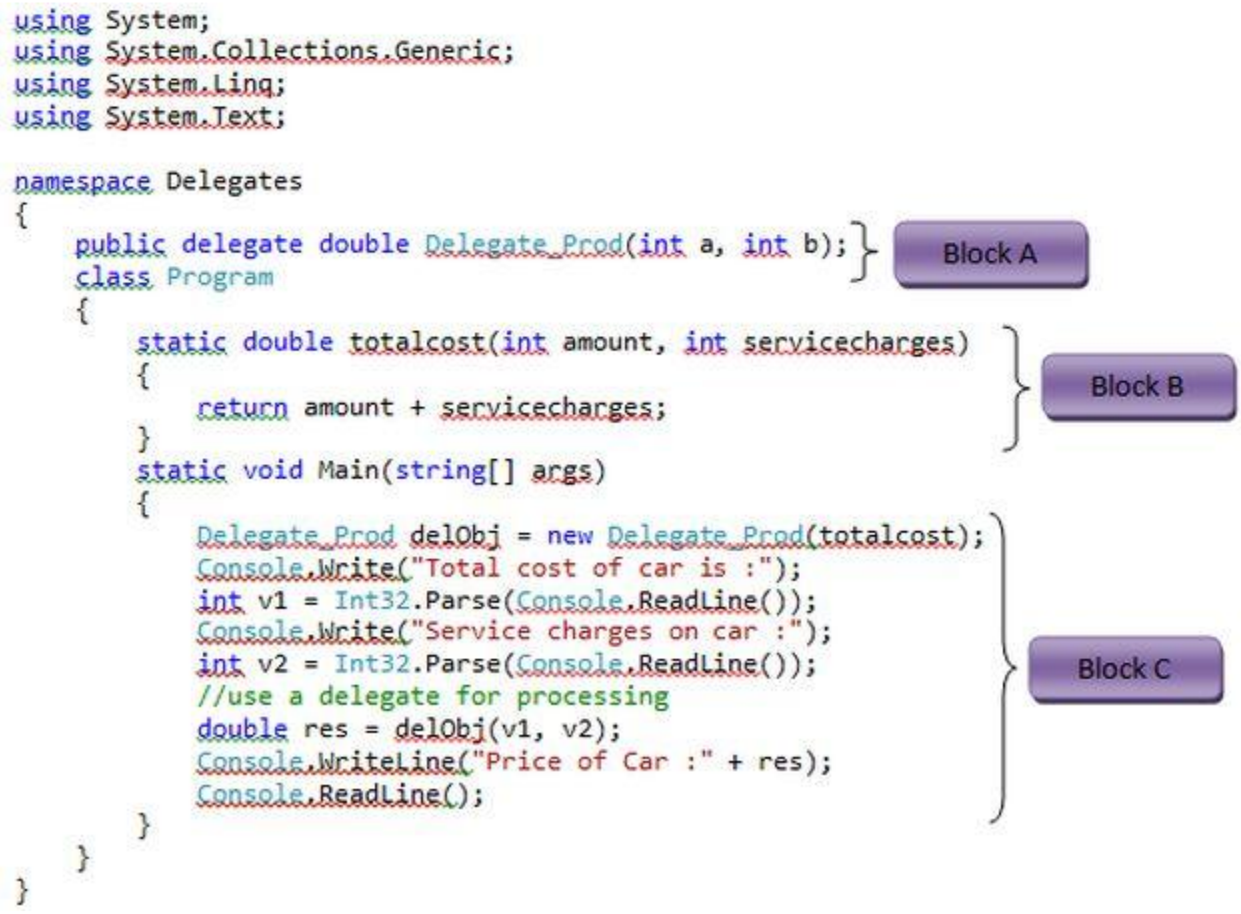
In simple words, a delegate is a type that safely encapsulates a method; a delegate is a pointer to a method. Just like you can pass a variable by reference, you can pass a reference to a method. Delegates are often used to implement callbacks and event listeners. A delegate does not need to know anything about the classes or methods it works with.

A Delegate consists of the following three parts:

- Declare a Delegate
- Instantiate the Delegate

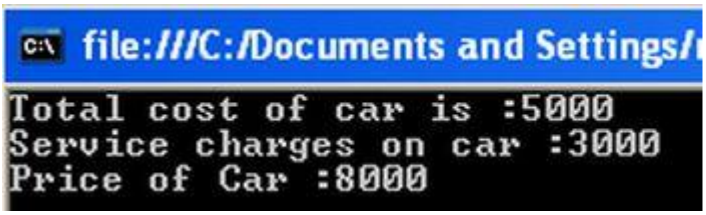
- Execute the Delegate

Simple Example 1:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Delegates
{
    public delegate double Delegate_Prod(int a, int b);
    class Program
    {
        static double totalcost(int amount, int servicecharges)
        {
            return amount + servicecharges;
        }
        static void Main(string[] args)
        {
            Delegate_Prod delObj = new Delegate_Prod(totalcost);
            Console.Write("Total cost of car is :");
            int v1 = Int32.Parse(Console.ReadLine());
            Console.Write("Service charges on car :");
            int v2 = Int32.Parse(Console.ReadLine());
            //use a delegate for processing
            double res = delObj(v1, v2);
            Console.WriteLine("Price of Car :" + res);
            Console.ReadLine();
        }
    }
}
```

Output:



```
Total cost of car is :5000
Service charges on car :3000
Price of Car :8000
```

Block A:

- In this sessoion we have defined a delegate with the name Delegate_Pro with two parameters.

Block B:

- In this sessoion we have defined the method totalcost with parameters.
- In the implementation of that method we have the sum the two values and return that.

Block C:

- First we have defined the object of the delegate passing method as the parameter.
- Then we will input two parameters from the user and save them in two variables.

- Now we will activate the delegate using the object we created and save the result in another variable.
- In the end we will show the result in the console.

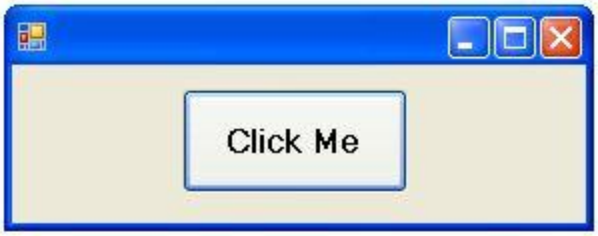
Events

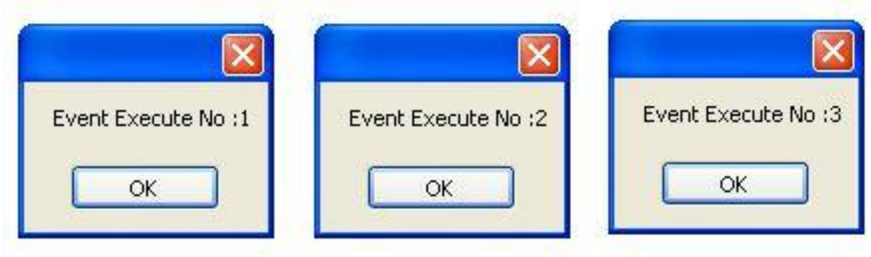
A hook on an object where the code outside of the object can say "When that something happens, that fires this event, please call my code".

Simple Example 2:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace @event
{
    public partial class Form1 : Form
     {
        public int a=1;

        public Form1()
        {
                InitializeComponent();
                this.button1.Click += new System.EventHandler(this.button1_Click);
                this.button1.Click += new System.EventHandler(this.button1_Click);

        }

        private void button1_Click(object sender, EventArgs e)
        {

            MessageBox.Show(a.ToString());
            a++;
        }
    }
}
```

Output:




Event Execute No :1

Event Execute No :2

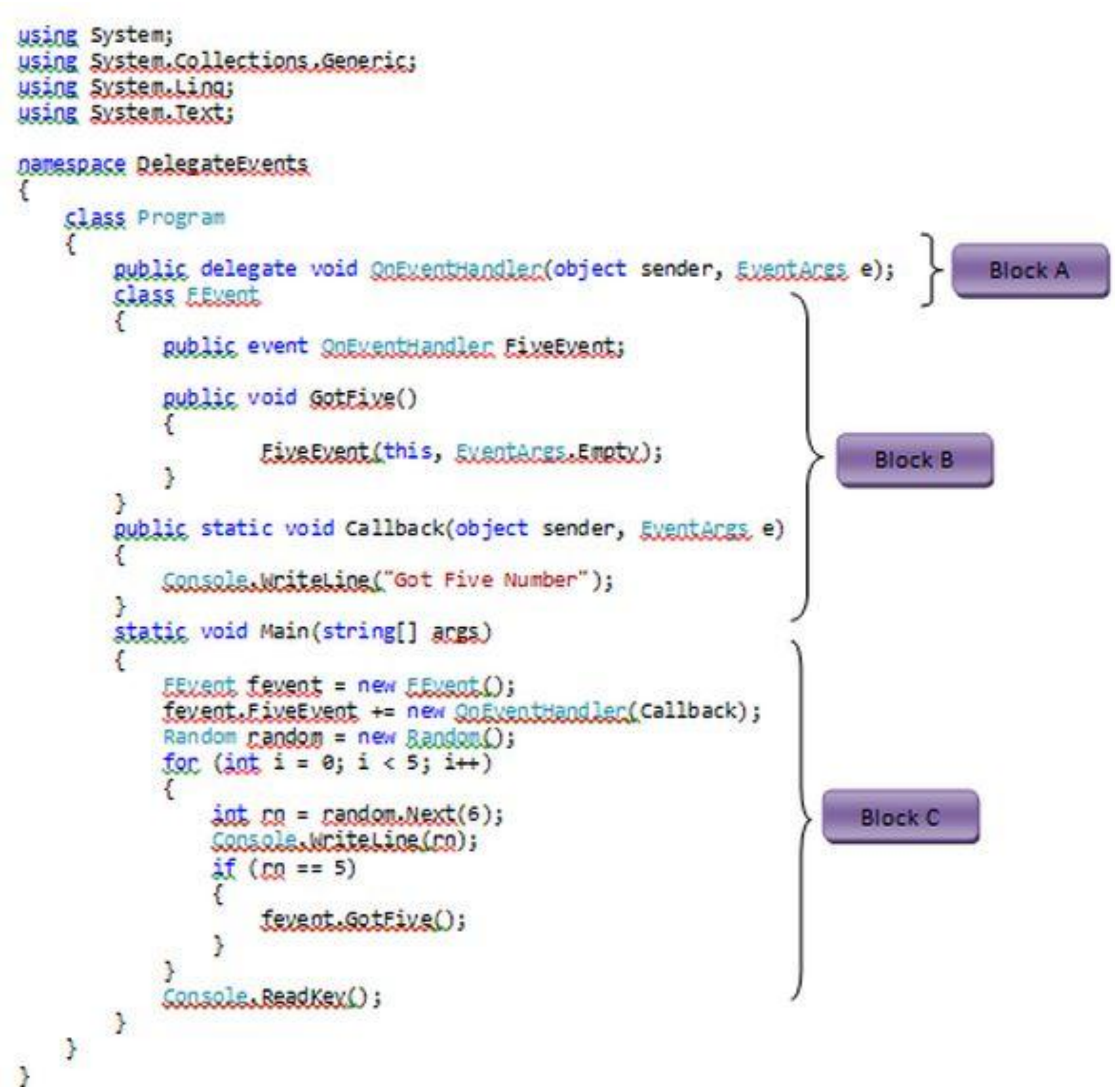
Event Execute No :3

In the preceding example we have defined the simple event on button click. When we click the button once, it will display the message box three times or call the click event three times.

Delegate and Events

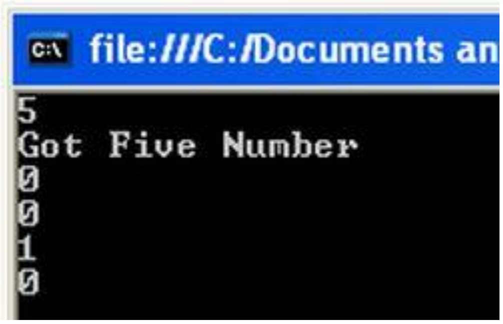
The best uses of delegates are in the events and we have daily use delegates with events without knowing them.

Simple Example 3:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace DelegateEvents
{
    class Program
    {
        public delegate void OnEventHandler(object sender, EventArgs e);
        class FEvent
        {
            public event OnEventHandler FiveEvent;

            public void GotFive()
            {
                    FiveEvent(this, EventArgs.Empty);
            }
        }
        public static void Callback(object sender, EventArgs e)
        {
            Console.WriteLine("Got Five Number");
        }
        static void Main(string[] args)
        {
            FEvent fevent = new FEvent();
            fevent.FiveEvent += new OnEventHandler(Callback);
            Random random = new Random();
            for (int i = 0; i < 5; i++)
            {
                int rn = random.Next(6);
                Console.WriteLine(rn);
                if (rn == 5)
                {
                    fevent.GotFive();
                }
            }
            Console.ReadKey();
        }
    }
}
```

Output:



```
5
Got Five Number
0
0
1
0
```

Block A:

- In this session we have defined a delegate with the name OnEventHandler with two parameters.

Block B:

- In this session we have tagged the delegate with the event FiveEvent.
- We have defined the method Gotfive. In that method we have activated the delegate using FiveEvent(this, EventArgs.Empty);
- Another method callback having the same signature with delegates.

Block C:

- In this sessoion we have defined an object of the class FEvent that is fevent.
- Using that fevent we have instantiated the Delegate with the callback method.
- Create the object of the Random class with the name of random.
- We use a for loop and generate five numbers using the random object.
- It will check the numbers; for each one that's five, it will activate the delegate.

Thank You I hope you like this E-Book.